Nordlys Documentation

Release 0.2

IAI group - University of Stavanger

Mar 10, 2021

Contents

Nordlys is a toolkit for entity-oriented and semantic search, created by the IAI group at the University of Stavanger.

Entities (such as people, organizations, or products) are meaningful units for organizing information and can provide direct answers to many search queries. Nordlys is a toolkit for entity-oriented and semantic search.

Nordlys supports 3 functionalities in the context of entity-oriented search:

- Entity retrieval: Returns a ranked list of entities in response to a query
- Entity linking: Identifies entities in a query and links them to the corresponding entry in the Knowledge base
- Target type identification: Detects the target types (or categories) of a query

Check our *Web interface documentation* for illustration of each of these functionalities.

CHAPTER 1

Nordlys can be used . . .

- through a *web-based GUI*
- through a *RESTful API*
- as a *command line tool*
- as a *Python package*

CHAPTER 2

Contents

2.1 Installation

Nordlys is a general-purpose semantic search toolkit, which can be used either as a Python package, as a command line tool, or as a service.

Data are a first-class citizen in Nordlys. To make use of the full functionality, the required data backend (MongoDB and Elasticsearch) need to be set up and data collections need to be loaded into them. There is built-in support for specific data collections, including DBpedia and Freebase. You may use the data dumps we prepared, or download, process and index these datasets from the raw sources.

2.1.1 1 Installing the Nordlys package

This step is required for all usages of Nordlys (i.e., either as a Python package, command line tool or service).

1.1 Environment

Nordlys requires Python 3.5+ and a Python environment you can install packages in. We highly recommend using an Anaconda Python distribution.

1.2 Obtain source code

You can clone the Nordlys repo using the following:

```
$ git clone https://github.com/iai-group/nordlys.git
```

1.3 Install prerequisites

Install Nordlys prerequisites using pip:

```
$ pip install -r requirements.txt
```

If you don't have pip yet, install it using

```
$ easy_install pip
```

Note: On Ubuntu, you might need to install lxml using a package manager

```
$ apt-get install python-lxml
```

1.4 Test installation

You may check if your installation has been successful by running any of the cmd_usage, e.g., from the root of your Nordlys folder issue

```
$ python -m nordlys.core.retrieval.retrieval
```

Alternatively, you can try importing Nordlys into a Python project. Make sure your local Nordlys copy is on the `PYTHONPATH`. Then, you may try, e.g., `from nordlys.core.retrieval.scorer import Scorer`.

Mind that this step is only to check the Python dependencies. It is rather limited what you can do with Nordlys without setting up data backed and loading the data components.

2.1.2 2 Setting up data backend

We use MongoDB and Elasticsearch for storing and indexing data. You can either connect to these services already running on some server or set these up on your local machine.

2.1 MongoDB

If you need to install MongoDB yourself, follow the instructions here.

Adjust the settings in `config/mongo.json`, if needed.

If you're using macOS, you'll likely need to change the soft limit of maxfiles to at least 64000, for mongoDB to work properly. Check the maxfiles limit of your system using:

```
$ launchctl limit
```

2.2 Elasticsearch

If you need to install Elasticsearch yourself, follow the instructions here. Note that Elasticsearch requires Java version 8.

Adjust the settings in `config/elastic.json`, if needed.

Nordlys has been tested with Elasticsearch version 5.x (and would likely need updating for newer version).

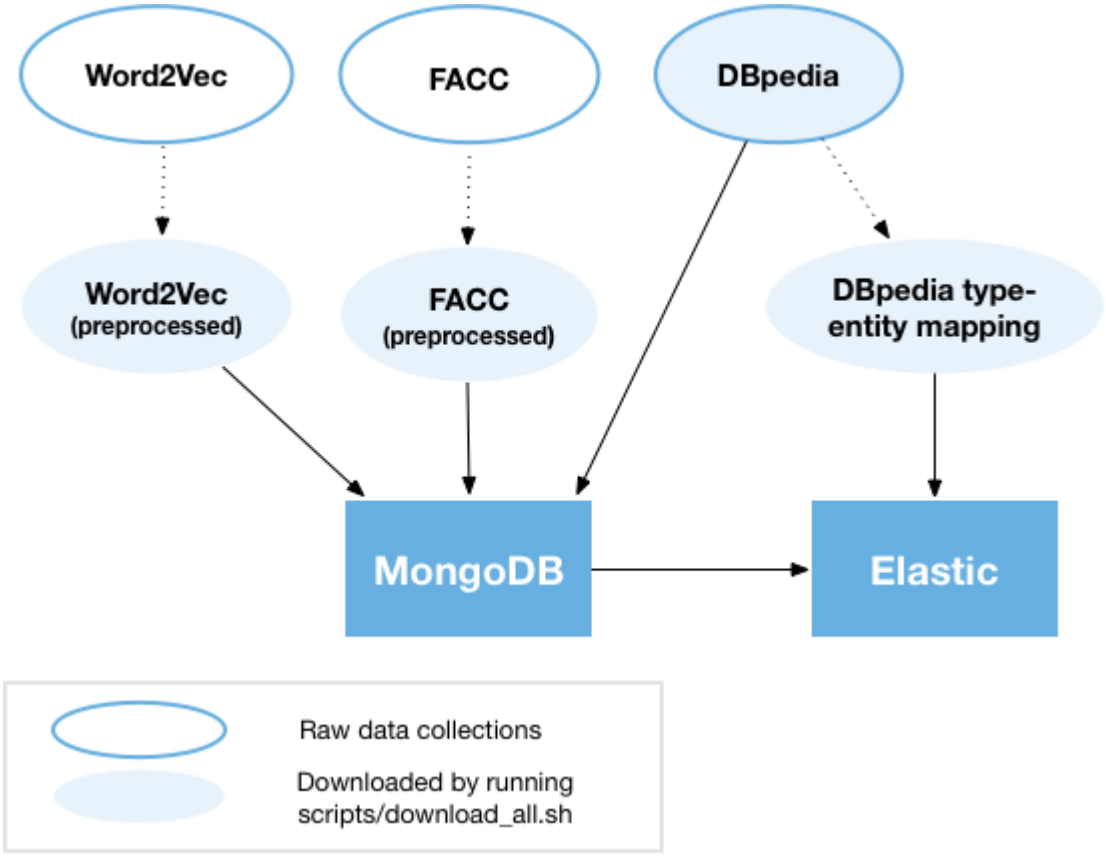
2.1.3 3 Loading data components

Data are a crucial component of Nordlys. While most of the functionality is agnostic of the underlying knowledge base, there is built-in support for working with specific data sources. This primarily means DBpedia, with associated resources from Freebase.

Specifically, Nordlys is shipped with functionality designed around **DBpedia 2015-10**, and the dumps we provide are for that particular version. However, we provide config files and instructions for working with **DBpedia 2016-10**, should you prefer a newer version.

Note that you may need only a certain subset of the data, depending on the required functionality. See *Data components* for a detailed description.

The figure below shows an overview of data sources and their dependencies.



3.1 Load data to MongoDB

You can either load the data to MongoDB (i) from dumps that we made available or (ii) from the raw source files (DBpedia, FACC, Word2vec, etc.). Below, we discuss the former option. For the latter, see *Building MongoDB sources from raw data*. Note that processing from the raw sources takes significantly longer because of the nontrivial amount of data.

To load the data to MongoDB, you need to run the following commands from the main Nordlys folder. Note that the first dump is required for the core Nordlys functionality over DBpedia. The other dumps are optional, depending on whether the respective functionality is needed.

Command	Required for
`./scripts/load_mongo_dumps.sh mongo_dbpedia-2015-10.tar.bz2`	All

`./scripts/load_mongo_dumps.sh mongo_surface_forms_dbpedia.tar.bz2`
`./scripts/load_mongo_dumps.sh mongo_surface_forms_facc.tar.bz2`
`./scripts/load_mongo_dumps.sh mongo_fb2dbp-2015-10.tar.bz2`
	EL and EC

`./scripts/load_mongo_dumps.sh mongo_word2vec-googlenews.tar.bz2`	TTI

3.2 Download auxiliary data files

The following files are needed for various services. You may download them all using

```
$ ./scripts/download_auxiliary.sh
```

Description	Location (relative to main Nordlys folder)	Required for
Type-to-entity mapping	`data/raw-data/dbpedia-2015-10/type2entity-mapping`	TTI
Freebase-to-DBpedia mapping	`data/raw-data/dbpedia-2015-10/freebase2dbpedia`	EL
Entity snapshot	`data/el`	EL [1]

- [1] If entity annotations are to be limited to a specific set; this file contains the proper named entities in DBpedia 2015-10

3.3 Build Elastic indices

There are multiple elastic_indices created for supporting different services. Run the following commands from the main Nordlys folder to build the indices for the respective functionality.

Command	Source	Required for
`./scripts/build_dbpedia_index.sh core`	MongoDB	ER, EL, TTI
`./scripts/build_dbpedia_index.sh types`	Raw DBpedia files [1]	TTI
`./scripts/build_dbpedia_index.sh uri`	MongoDB	ER [2]

- [1] Requires short entity abstracts and instance types files
- [2] only for ELR model

Note: To use the 2016-10 version of DBpedia, add `2016-10` as a 2nd argument to the above scripts.

2.2 Data components

2.2.1 Data sources

DBpedia

We use DBpedia as the main underlying knowledge base. In particular, we prepared dumps for DBpedia version 2015-10.

DBpedia is distributed, among other formats, as a set of .ttl.bz2 files. We use a selection of these files, as defined in `data/config/dbpedia2mongo.config.json`. You can download these files from DBpedia Website directly or running `./scripts/download_dbpedia.sh` from the main Nordlys folder. Running the script will place the downloaded files under `data/raw-data/dbpedia-2015-10/`.

We also provide a minimal sample from DBpedia under `data/dbpedia-2015-10-sample`, which can be used for testing/development in a local environment.

FACC

The Freebase Annotations of the ClueWeb Corpora (FACC) is used for building entity surface form dictionary. You can download the collection from its main Website. and further process it using our scripts. Alternatively, you can download the preprocessed data from our server. Check the README file under *data/raw-data/facc* for detailed information.

Word2Vec

Word2Vec vectors (300D) trained on Google News corpus, which canbe dowloaded from the its Website. Check the README file under *data/raw-data/word2vec* for detailed information.

2.2.2 MongoDB collections

The table below provides an overview of the MongoDB collections that are used by the different services.

Name	Description	EC	ER	EL	TTI
`dbpedia-2015-10`	DBpedia	+[1]	+[2]		+[3]
`fb2dbp-2015-10`	Mapping from Freebase to DBpedia IDs	+[4]		+	
`surface_forms_dbpedia`	Entity surface forms from DBpedia	+[5]		+[6]	
`surface_forms_facc`	Entity surface forms from FACC	+[7]		+	
`word2vec-googlenews`	Word2vec trained on Google News				+[8]

- [1] for entity ID-based lookup and DBpedia2Freebase mapping functionalities
- [2] only for building the Elastic entity index; not used later in the retrieval process
- [3] for entity-centric TTI method
- [4] for Freebase2DBpedia mapping functionality
- [5] for entity surface form lookup from DBpedia
- [6] for all EL methods other than "commonness"
- [7] for entity surface form lookup from FACC
- [8] for LTR TTI method

Building MongoDB sources from raw data

To build the above tables from raw data (as opposed to the provided dumps), first make sure that you have the raw data files.

- For DBpedia, these may be downloaded using `./scripts/download_dbpedia.sh`
- For the FACC and Word2vec data files, execute `./scripts/download_raw.sh`

To load DBpedia to MongoDB, run

```
python -m nordlys.core.data.dbpedia.dbpedia2mongo data/config/dbpedia-2015-10/
↪dbpedia2mongo.config.json
```

Note: To use the DBpedia 2015-10 sample shipped with Nordlys, as opposed to the full collection, change the value `path` to `data/raw-data/dbpedia-2015-10_sample/` in `dbpedia2mongo.config.json`.

2.2.3 Elastic indices

Name	Description	ER	EL	TTI
dbpedia_2015_10	DBpedia index	+	+[1]	+[2]
dbpedia_2015_10_uri	DBpedia URI-only index	+[3]		
dbpedia_2015_10_types	DBpedia types index			+[4]

- [1] for all EL methods other than "commonness"
- [2] only for entity-centric TTI method
- [3] only for ELR entity ranking method
- [4] only for type-centric TTI method

2.3 RESTful API

The following Nordlys services can be accessed through a RESTful API:

- *Entity Retrieval*
- *Entity Linking in Queries*
- *Target Type Identification*
- *Entity Catalog*

Below we describe the usage for each of the services.

The Nordlys endpoint URL is http://api.nordlys.cc/ .

2.3.1 Entity Retrieval

The service presents a ranked list of entities in response to an entity-bearing query.

Endpoint URI

```
http://api.nordlys.cc/er
```

Example

Request:

http://api.nordlys.cc/er?q=total+recall&1st_num_docs=100&model=lm

Response:

```
{"query": "total recall",}
  "total_hits": 1000",
  "results": {
    "0": {
      "entity": "<dbpedia:Total_Recall_(1990_film)>",
      "score": -10.042525028471253
    },
    "1": {
      "entity": "<dbpedia:Total_Recall_(2012_film)>",
      "score": -10.295316626850521
    },
    ...
}
```

Parameters

The following table lists the parameters needed in the request URL for entity retrieval.

Parameters	
q *(required)*	Search query
1st_num_docs	The number of documents that will be re-ranked using a model. The recommended value (esp. for baseline comparisons) is 1000. Lower values, like 100, are recommended only when efficiency matters *(default: 1000)*.
start	Starting offset for ranked documents *(default:0)*.
fields_return	Comma-separated list of fields to return for each hit *(default: "")*.
model	Name of the retrieval model; Accepted values: "bm25", "lm", "mlmprms" *(default: "lm")*. • BM25: The BM25 model, as implemented in Elasticsearch. This is the most efficient that is provided by Nordlys (to this date). • LM: Language Modeling [1] approach, which employs a single a single field representation of entities. • MLM: The Mixture of Language Models [2], which uses a linear combination of language models built for each field. • PRMS: The Probabilistic Model for Semistructured Data [3], which uses collection statistics to compute field weights in for MLM model.
field	The name of the field used for LM *(default: "catchall")*.
fields	Comma-separated list of the fields for PRMS *(default: "catchall")*.
field_weights	Comma-separated list of fields and their corresponding weights for MLM *(default: "catchall:1")*.
smoothing_method	Smoothing method for LM-based models; Accepted values: jm, dirichlet *(default: "dirichlet")*.
smoothing_param	The value of smoothing parameters (lambda or mu); Accepted values: float, "avg_len" *(default for "jm": 0.1, default for "dirichlet": 2000)*.

2.3.2 Entity Linking in Queries

The service identifies entities in queries and links them to the corresponding entry in the Knowledge base (DBpedia).

Endpoint URI

```
http://api.nordlys.cc/el
```

Example

- **Request:** http://api.nordlys.cc/el?q=total+recall
- **Response:**

```
{
  "processed_query": "total recall",
  "query": "total recall",
  "results": [
    {
      "entity": "<dbpedia:Total_Recall_(1990_film)>",
      "mention": "total recall",
      "score": 0.4013333333333334
    },
    {
      "entity": "<dbpedia:Total_Recall_(2012_film)>",
      "mention": "total recall",
      "score": 0.315
    }
  ]
}
```


Parameters

The following table lists the parameters needed in the request URL for entity linking.

Parameters	
q *(required)*	The search query
method	The name of method; Accepted values *(default: "cmns")* • cmns: The baseline method that uses the overall. popularity of entities as link targets, implemented based on [5]. • ltr: The learning-to-rank model, implemented based on the LTR-greedy in [9]. Note that the implemented method is slightly different from [9] (due to efficiency reasons).
threshold	The entity linking threshold *(default: 0.1)*.

2.3.3 Target Type Identification

The service assigns target types (or categories) to queries from the DBpedia type taxonomy.

Endpoint URI

```
http://api.nordlys.cc/tti
```

Example

- **Request:** http://api.nordlys.cc/tti?q=obama
- **Response:**

```
{
  "query": "obama",
  "results": {
    "0": {
      "score": 3.3290777,
      "type": "<dbo:Ambassador>"
    },
    "1": {
      "score": 3.2955842,
      "type": "<dbo:Election>"
    },
    ...
}
```


Parameters

The following table lists the parameters needed in the request URL for target type identification.

Parameters	
q *(required)*	The search query
method	The name of method; accepted values: "tc", "ec", "ltr" *(default: "tc")*. • TC: The Type Centric (TC) method based on [6]. Both BM25 and LM models can be used as a retrieval model here. This method fits the early fusion design pattern in [7]. • EC: The Entity Centric (EC) method, as described in [6]. Both BM25 and LM models can be used as a retrieval model here. This method fits the late fusion design pattern in [7]. • LTR: The Learing-To-Rank (LTR) method, as proposed in [8].
num_docs	The number of top ranked target types to retrieve *(default: 10)*.
start	The starting offset for ranked types.
model	Retrieval model, if method is "tc" or "ec"; Accepted values: "lm", "bm25".
ec_cutoff	If method is "ec", rank cut-off of top-K entities for EC TTI.
field	Field name, if method is "tc" or "ec".
smoothing_method	If model is "lm", smoothing method; accepted values: "jm", "dirichlet".
Smoothing_param	If model is "lm", smoothing parameter; accepted values: float, "avg_len".

2.3.4 Entity Catalog

This service is used for representing entities (with IDs, name variants, attributes, and relationships). Additionally, it provides statistics that can be utilized, among others, for result presentation (e.g., identifying prominent properties when generating entity cards).

Endpoint URI

```
http://api.nordlys.cc/ec
```

Look up entity by ID

- **Request:**

```
http://api.nordlys.cc/ec/lookup_id/<entity_id>
```

- **Example:**

  http://api.nordlys.cc/ec/lookup_id/<dbpedia:Albert_Einstein>

- **Response:**

```
{
    "<dbo:abstract>": ["Albert Einstein was a German-born theoretical physicist ..
↪. ],
    "<dbo:academicAdvisor>": ["<dbpedia:Heinrich_Friedrich_Weber>"],
    "<dbo:almaMater>": [
        "<dbpedia:ETH_Zurich>",
        "<dbpedia:University_of_Zurich>"
                    ],
    "<dbo:award>": [
        "<dbpedia:Nobel_Prize_in_Physics>",
        "<dbpedia:Max_Planck_Medal>",
        ...
                ],
    "<dbo:birthDate>": ["1879-03-14"],
    ...
}
```

Look up entity by name (DBpedia)

Looks up an entity by its surface form in DBpedia.

- **Request:**

```
http://api.nordlys.cc/ec/lookup_sf/dbpedia/<sf>
```

- **Example:**

  http://api.nordlys.cc/ec/lookup_sf/dbpedia/new%20york

- **Response:**

```
{
    "_id": "new york"
    "<rdfs:label>" : {
      "<dbpedia:New_York>": 1
    }
    "<dbo:wikiPageDisambiguates>": {
      "<dbpedia:Manhattan>": 1,
      "<dbpedia:New_York,_Kentucky>": 1,
      ...
```

(continues on next page)

(continued from previous page)

```
    }
    ...
}
```

Look up entity by name (FACC)

Looks up an entity by its surface form in FACC.

- **Request:**

```
http://api.nordlys.cc/ec/lookup_sf/facc/<sf>
```

- **Example:**

  http://api.nordlys.cc/ec/lookup_sf/facc/new%20york

- **Response:**

```
{
    "_id" : "new york",
    "facc12" : {
      "<fb:m.02_286>": 18706787,
      "<fb:m.02_53fb>": 49,
      "<fb:m.02_b9l>": 87,
      "<fb:m.02_l43>": 12,
      "<fb:m.02_l5n>": 23963,
      ...
    }
}
```

Map Freebase entity ID to DBpedia ID

- **Request:**

```
http://api.nordlys.cc/ec/freebase2dbpedia/<fb_id>
```

- **Example:**

  http://api.nordlys.cc/ec/freebase2dbpedia/<fb:m.02_286>

- **Response:**

```
{
  "dbpedia_ids": [
    "<dbpedia:New_York_City>"
  ]
}
```

Map DBpedia entity ID to Freebase ID

- **Request:**

```
http://api.nordlys.cc/ec/dbpedia2freebase/<dbp_id>
```

- **Example:**

  http://api.nordlys.cc/ec/dbpedia2freebase/<dbpedia:New_York>

- **Response:**

```
{
  "freebase_ids": [
    "<fb:m.059rby>"
  ]
}
```

Parameters

The following table lists the parameters needed in the request URL for entity catalog.

Parameter	
entity id	It is in the form of "<dbpedia:XXX>", where XXX denotes the DBpedia/Wikipedia ID of an entity.
sf	Entity surface form (e.g., "john smith", "new york"). It needs to be url-escaped.
fb_id	Freebase ID
bdp_id	DBpedia ID

2.3.5 References

[1] Jay M Ponte and W Bruce Croft . 1998. *A Language modeling approach to information retrieval*. In Proc. of SIGIR '98. 275–281.

[2] Paul Ogilvie and Jamie Callan. 2003. *Combining document representations for known-item search*. Proc. of SIGIR '03 (2003), 143–150.

[3] Jinyoung Kim, Xiaobing Xue, and W Bruce Croft . 2009. *A probabilistic retrieval model for semistructured data*. In Proc. of ECIR '09. 228–239.

[4] Faegheh Hasibi, Krisztian Balog, and Svein Erik Bratsberg. 2016. *Exploiting entity linking in queries for entity retrieval*. In Proc. of ICTIR '16. 171–180. [BIB] [PDF]

[5] Faegheh Hasibi, Krisztian Balog, and Svein Erik Bratsberg. 2015. *Entity linking in queries: Tasks and Evaluation*. In Proc. of ICTIR '15. 171–180. [BIB] [PDF]

[6] Krisztian Balog, Robert Neumayer. 2012. *Hierarchical target type identification for entity-oriented queries*. In Proc. of CIKM '12. 2391–2394. [BIB] [PDF]

[7] Shuo Zhang and Krisztian Balog. *Design Patterns for Fusion-Based Object Retrieval*. In Proc. of ECIR '17. 684-690. [BIB] [PDF]

[8] Darío Garigliotti, Faegheh Hasibi, and Krisztian Balog. *Target Type Identification for Entity-Bearing Queries*. In Proc. of SIGIR '17. [BIB] [PDF]

[9] Faegheh Hasibi, Krisztian Balog, and Svein Erik Bratsberg. 2017. *Entity linking in queries: Efficiency vs. Effectiveness*. In Proc. of ECIR '17. 40-53. [BIB] [PDF]

2.4 Web-based GUI

Nordlys is shipped with a web-based graphical user interface, which is built on the Nordlys API. It is a wrapper for all functionalities provided by Nordlys toolkit and can be used, e.g., to perform user studies on result presentation.

The implementation of the Web interface is based on Flask and Bootstrap. Below we describe the functionalities provided by the excerpts from the Web GUI with their interface.

Note: The interface can be accessed via: http://gui.nordlys.cc/

2.4.1 Entity search

The entity search tab provides a ranked list of entities in response to an entity-bearing query. We generate the results by calling the Entity retrieval service of our API; e.g.,

```
http://api.nordlys.cc/er?q=total+recall&model=lm&1st_num_docs=100&fields_
↪return=abstract
```

By clicking on each result, we present an entity card, containing the factual summary of the entity and an image. We use the entity catalog (EC) service to generate these cards.

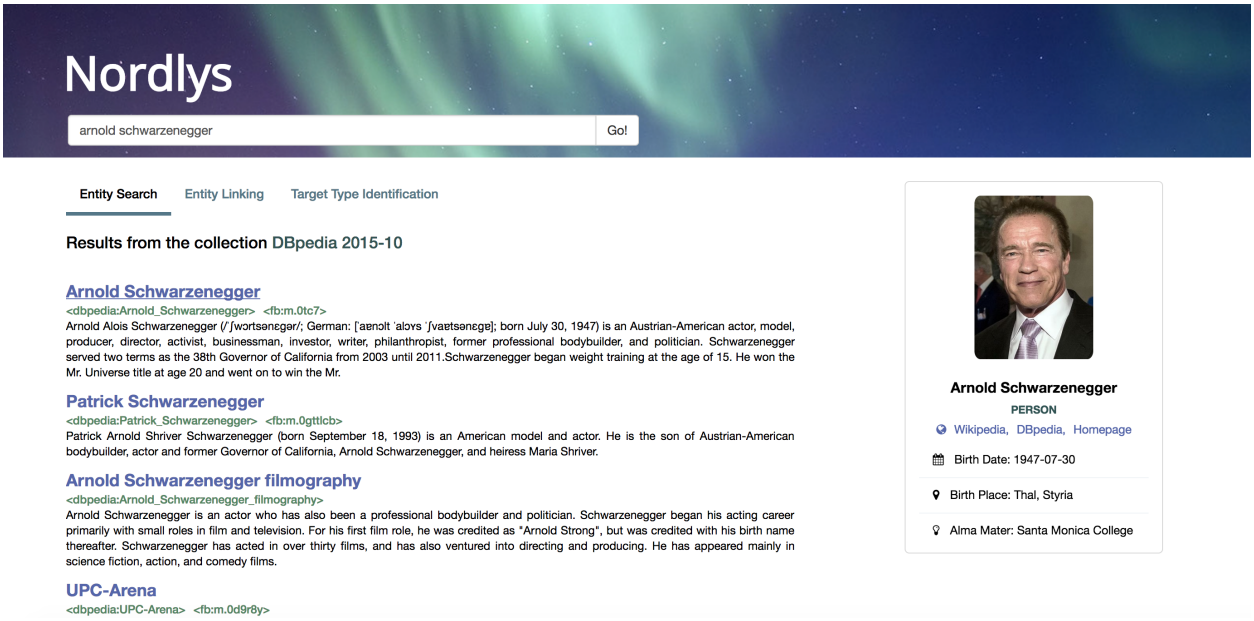


Fig. 1: Nordlys Web interface - Entity Search

2.4.2 Entity linking in queries

For entity linking in queries, we use the baseline CMNS method, with threshold 0.1. For example, we make the following call to the entity linking service of our API:

```
http://api.nordlys.cc/el?q=arnold+schwarzenegger+total+recall&method=cmns&threshold=0.
↪1
```

2.4.3 Target Type Identification

For target type identification, we employ the type centric method; e.g.,

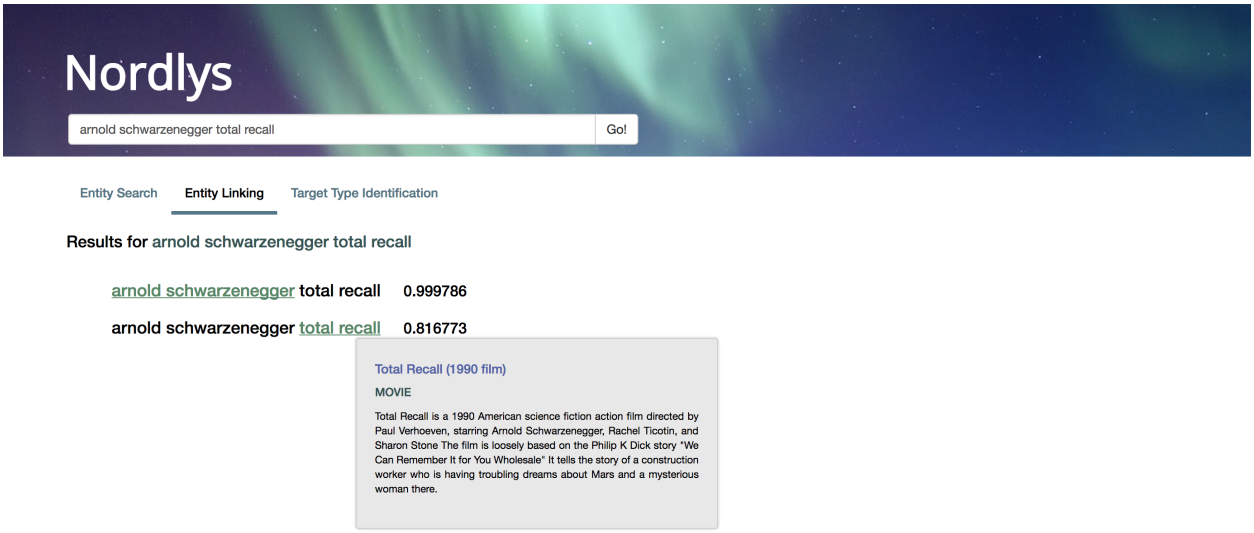


Fig. 2: Nordlys Web interface - Entity Linking

```
http://api.nordlys.cc/tti?q=obama
```

Note: For detailed information about our API calls, see api_usage

2.5 Command line usage

Nodlys provides the following command line applications:

- **Core applications**
  - Retrieval *general usage* and *configurations*
  - Machine learning *general usage* and configurations
- **Entity-oriented applications**
  - *Entity catalog*
  - *Entity retrieval*
  - Entity linking
  - *Target type identification*

2.6 Nordlys architecture

Nordlys is based on a multitier architecture with three layers:

- *core* (data tier)
- *logic*

Fig. 3: Nordlys Web interface - Target Type Identification

- *services* (presentation tier)

2.6.1 Core tier

The *core* tier provides basic functionalities and is conneted to various the third-party tools. The functionalities include:

- *Retrieval* (based on Elasticsearch)
- *Storage* (based on MongoDB)
- *Machine learning* (based on scikit-learn)
- *Evaluation* (based on trec-eval)

Additionally, a separate *data* package is provided with functionality for loading and preprocessing standard data sets (DBpedia, Freebase, ClueWeb, etc.).

It is possible to connect additional external tools (or replace our default choices) by implementing standard interfaces of the respective core modules.

Note: The core layer represents a versatile general-purpose modern IR library, which may also be accessed using command line tools.

2.6.2 Logic tier

The *logic* tier contains the main business logic, which is organized around five main modules:

1. *Entity* provides access to the entity catalog (including knowledge bases and entity surface form dictionaries.
2. *Query* provides the representation of search queries along with various preprocessing methods.
3. *Features* is a collection of entity-related features, which may be used across different search tasks.

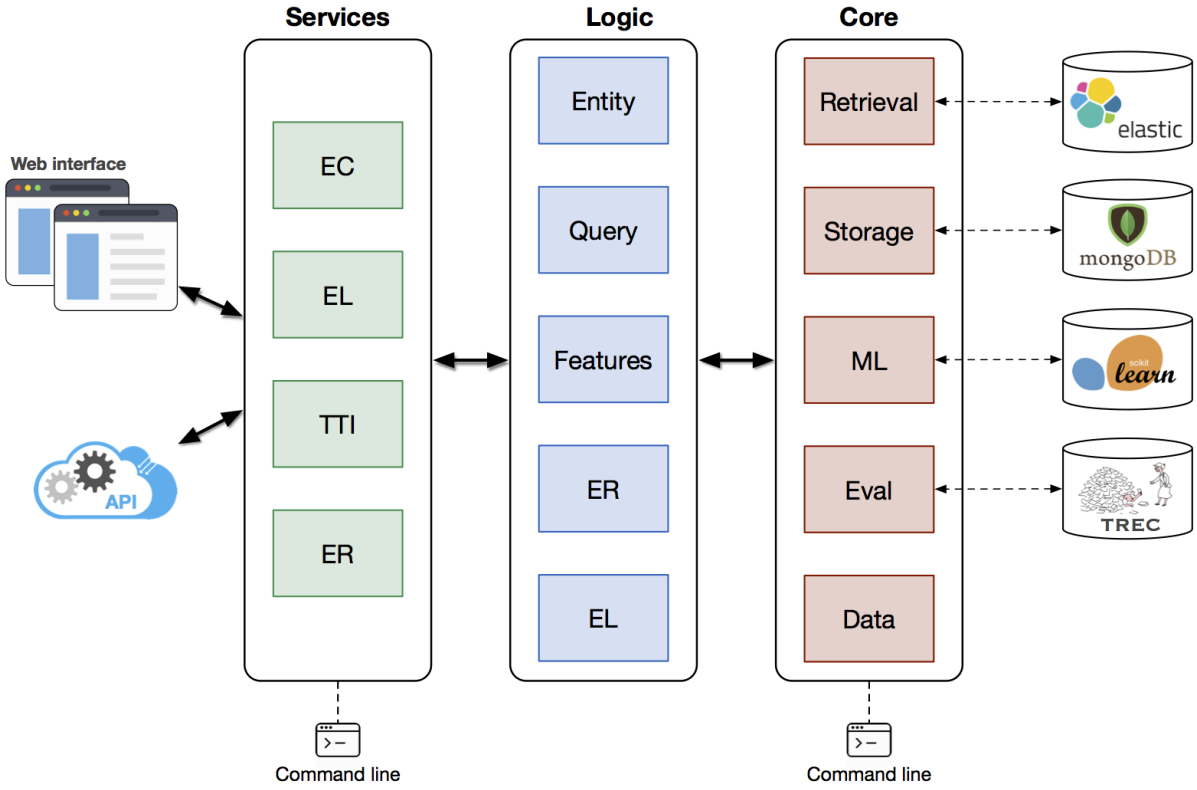


Fig. 4: Nordlys architecture.

4. *Entity retrieval* contains various entity ranking methods.
5. *Entity linking* implements entity linking functionality.

The logic layer may not be accessed directly (i.e.,as a service or as a command line application).

2.6.3 Services tier

The *services* tier provides end-user access to the toolkit's functionality, throughout the *command line*, *API*, and *web interface*. Four main types of service is available:

1. *Entity retrieval*
2. *Entity linking*
3. *Target type identification*
4. *Entity catalog*

2.7 nordlys package

Nordlys toolkit is provided as the `nordlys` Python package.

Nordlys is delivered with an extensive and detailed documentation, from package-level overviews, to module usage examples, to fully documented classes and methods.

2.7.1 Subpackages

nordlys.core package

Core packages

These low-level packages (basic IR, ML, NLP, storage, etc.) are basic building blocks that do not have any interdependencies among each other.

Subpackages

nordlys.core.data package

Subpackages

nordlys.core.data.dbpedia package

Submodules

nordlys.core.data.dbpedia.create_sample module

nordlys.core.data.dbpedia.dbpedia2mongo module

nordlys.core.data.dbpedia.dbpedia_surfaceforms2mongo module

nordlys.core.data.dbpedia.freebase2dbpedia2mongo module

nordlys.core.data.dbpedia.indexer_dbpedia module

nordlys.core.data.dbpedia.indexer_dbpedia_types module

DBpedia Types Indexer

Builds a DBpedia type index from entity abstracts.

The index is build directly from DBpedia files in .ttl.bz2 format (i.e., MongoDB is not needed).

Usage

```
python -m nordlys.core.dbpedia.indexer_dbpedia_types -c <config_file>
```

Config parameters

- **index_name**: name of the index
- **dbpedia_files_path**: path to DBpedia .ttl.bz2 files

Authors Krisztian Balog, Dario Garigliotti

class nordlys.core.data.dbpedia.indexer_dbpedia_types.**IndexerDBpediaTypes**(*config*)
Bases: object

build_index(*force=False*)
Builds the index.

Note: since DBpedia only has a few hundred types, no bulk indexing is needed.

Parameters **force** – True iff it is required to overwrite the index (i.e. by creating it by force); False by default. :type force: bool :return:

name

nordlys.core.data.dbpedia.indexer_dbpedia_types.**arg_parser**()

nordlys.core.data.dbpedia.indexer_dbpedia_types.**main**(*args*)

nordlys.core.data.dbpedia.indexer_dbpedia_uri module

nordlys.core.data.facc package

Submodules

nordlys.core.data.facc.facc2mongo module

Facc to Mongo

Adds entity surface forms from the Freebase Annotated ClueWeb Corpora (FACC).

The input to this script is (name variant, Freebase entity, count) triples. See *data/facc1/README.md* for the preparation of FACC data in such format.

Authors Krisztian Balog, Faegheh Hasibi

class nordlys.core.data.facc.facc2mongo.**FACCToMongo**(*config*)
Bases: object

Inserts FACC surface forms to Mongo.

build()
Builds surface form collection from FACC annotations.

nordlys.core.data.facc.facc2mongo.**arg_parser**()

nordlys.core.data.facc.facc2mongo.**main**(*args*)

nordlys.core.data.word2vec package

Submodules

nordlys.core.data.word2vec.word2vec2mongo module

Word2vec to Mongo

Loads Word2Vec to MongoDB.

Authors Faegheh Hasibi, Dario Garigliotti

class nordlys.core.data.word2vec.word2vec2mongo.**Word2VecToMongo**(*config*)
Bases: object

build()
Builds word2vec collection from GoogleNews 300-dim pre-trained corpus.

nordlys.core.data.word2vec.word2vec2mongo.**arg_parser**()

nordlys.core.data.word2vec.word2vec2mongo.**main**(*args*)

nordlys.core.eval package

This is the (**intro** of the) eval package doc. It's in the `__init__.py` of the package, as an example for the other packages, and IT SHOULD BE cleaned up without all of this ;).

It could contain some/all eval hints.

It might contain, e.g., items in this way

- This
- That

and also code snippets like usual, using as usual these double colons and the indented code:

```
instance = Instance(id)
```

And that's it.

(In the following I included all the up-to-now available modules, sectioned by criterion at first sight.)

Submodules

nordlys.core.eval.eval module

Evaluation

Console application for eval package.

Author Faegheh Hasibi

class nordlys.core.eval.eval.**Eval**(*operation*, *qrels=None*, *runs=None*, *metric=None*, *output_file=None*)

Bases: object

Main entry point for eval package.

Parameters

- **operation** – operation (see *OPERATIONS* for allowed values)
- **qrels** – name of qrels file
- **runs** – name of run files
- **metric** – metric

OPERATIONS = ['query_diff']

OP_QUERY_DIFF = 'query_diff'

run()

Runs the operation with the given arguments.

nordlys.core.eval.eval.**arg_parser**()

nordlys.core.eval.eval.**main**(*args*)

nordlys.core.eval.plot_diff module

Plot Differences

Plots a series of scores which represent differences.

Authors Shuo Zhang, Krisztian Balog

class nordlys.core.eval.plot_diff.**QueryDiff**

Bases: object

SCORES = [25, 20, 10, 5, 0, -1, -5, -10]

create_pdf(*diff_file*, *pdf_file*, *title=''*, *xlabel=''*, *ylabel=''*, *aspect_ratio='equal'*, *separator='\t'*)

Create bar plot for differences in pdf.

This function is used to load difference .csv file, create bar plot and store as a pfd file.

Pdf created and saved pdf file

make_plot()
Make a bar plot using SCORES

nordlys.core.eval.query_diff module

Query Differences

Computes query-level differences between two runs.

Authors Shuo Zhang, Krisztian Balog, Dario Garigliotti

class nordlys.core.eval.query_diff.**QueryDiff**(*run1_file*, *run2_file*, *qrels*, *metric*)
Bases: object

Parameters

- **run1_file** – name of run1 file (baseline)
- **run2_file** – name of run2 file (new method)
- **qrels** – name of qrels file
- **metric** – metric

Returns

dump_differences(*output_file*)
Outputs query-level differences between two methods into a tab-separated file.

The first method is considered the baseline, the differences are with respect to that. Output format: queryID res1 res2 diff(res2-res1)

nordlys.core.eval.trec_eval module

Trec Evaluation

Wrapper for trec_eval.

Authors Dario Garigliotti, Shuo Zhang

class nordlys.core.eval.trec_eval.**TrecEval**
Bases: object

Holds evaluation results obtained using trec_eval.

evaluate(*qrels_file*, *run_file*, *eval_file=None*)
Evaluates a runfile using trec_eval. Optionally writes evaluation output to file.

Parameters

- **qrels_file** – name of qrels file
- **run_file** – name of run file
- **eval_file** – name of evaluation output file

get_query_ids()
Returns the set of queryIDs for which we have results.

get_score(*query_id*, *metric*)
Returns the score for a given queryID and metric.

Parameters

- **query_id** – queryID
- **metric** – metric

Returns score (or None if not found)

load_results(*eval_file*)
Loads results from an existing evaluation file.

Parameters **eval_file** – name of evaluation file

nordlys.core.eval.trec_qrels module

Trec Qrels

Utility module for working with TREC qrels files.

Usage

Get statistics about a qrels file `trec_qrels <qrels_file> -o stat`

Filter qrels to contain only documents from a given set `trec_qrels <qrels_file> -o filter_docs -d <doc_ids_file> -f <output_file>`

Filter qrels to contain only queries from a given set `trec_qrels <qrels_file> -o filter_qs -q <query_ids_file> -f <output_file>`

Author Krisztian Balog

class nordlys.core.eval.trec_qrels.**TrecQrels**(*file_name=None*)
Bases: object

Represents relevance judments (TREC qrels).

filter_by_doc_ids(*doc_ids_file*, *output_file*)
Filters qrels for a set of selected docIDs and outputs the results to a file.

Parameters

- **doc_ids_file** – File with one docID per line
- **output_file** – Output file name

filter_by_query_ids(*query_ids_file*, *output_file*)
Filters qrels for a set of selected queryIDs and outputs the results to a file.

Parameters

- **query_ids_file** – File with one queryID per line
- **output_file** – Output file name

get_queries()
Returns the set of queries.

get_rel(*query_id*)
Returns relevance level for a given query.

Parameters **query_id** – queryID

Returns dict (docID as key and relevance as value) or None

load(*file_name*)
Loads qrels from file.

Parameters **file_name** – name of qrels file

num_rel(*query_id*, *min_rel=1*)
Returns the number of relevant results for a given query.

Parameters

- **query_id** – queryID
- **min_rel** – minimum relevance level

Returns number of relevant results

print_stat()
Prints simple statistics.

nordlys.core.eval.trec_qrels.**arg_parser**()

nordlys.core.eval.trec_qrels.**main**(*args*)

nordlys.core.eval.trec_run module

Trec run

Utility module for working with TREC runfiles.

Usage

Get statistics about a runfile `trec_run <run_file> -o stat`

Filter runfile to contain only documents from a given set `trec_run <run_file> -o filter -d <doc_ids_file> -f <output_file> -n <num_results>`

Authors Krisztian Balog, Dario Garigliotti

class nordlys.core.eval.trec_run.**TrecRun**(*file_name=None*, *normalize=False*, *remap_by_exp=False*, *run_id=None*)

Bases: object

Represents a TREC runfile.

Parameters

- **file_name** – name of the run file
- **normalize** – whether retrieval scores are to be normalized for each query (default: False)
- **remap_by_exp** – whether scores are to be converted from the log-domain by taking their exp (default: False)

filter(*doc_ids_file*, *output_file*, *num_results=100*)
Filters runfile to include only selected docIDs and outputs the results to a file.

Parameters

- **doc_ids_file** – file with one doc_id per line

- **output_file** – output file name
- **num_results** – number of results per query

get_query_results(*query_id*)
Returns the corresponding RetrievalResults object for a given query.

Parameters **query_id** – queryID

Return type *nordlys.core.retrieval.retrieval_results.RetrievalResults*

get_results()
Returns all results.

Returns a dict with queryIDs as keys and RetrievalResults object as values

load_file(*file_name*, *remap_by_exp=False*)
Loads a TREC runfile.

Parameters

- **file_name** – name of the run file
- **remap_by_exp** – whether scores are to be converted from the log-domain by taking their exp (default: False)

normalize()
Normalizes the retrieval scores such that they sum up to one for each query.

print_stat()
Prints simple statistics.

nordlys.core.eval.trec_run.**arg_parser**()

nordlys.core.eval.trec_run.**main**(*args*)

nordlys.core.ml package

Machine learning

The machine learning package is connected to Scikit-learn and can be used for learning-to-rank and classification purposes.

Usage

For information on how to use the package from command line and set the configuration, read ML usage

Data format

The file format of the training and test files is *json*. Each instance is presented as a dictionary, consist of the following elements:

- **ID**: Instance id
- **Target**: The target value of the instance.
- **Features**: All the features, presented in key-value format. Note that all the instances should have the same set of features.

- **Properties**: All meta-data about the instance; e.g. query ID, or content.

Below is an excerpt from a json data file:

```
{
    "0": {
        "properties": {
            "query": "papaqui soccer",
            "entity": "<dbpedia:Soccer_(1985_video_game)>"
        },
        "target": "0",
        "features": {
            "feat1": 1,
            "feat2": 0,
            "feat3": 25
        }
    },
    "1": {}
}
```

Note: The sample files for using the ML package are provided under `data/ml_sample/` folder.

Note: Currently we provide support for Random Forest(RF) and Gradient Boosted Regression Trees (GBRT).

Submodules

nordlys.core.ml.cross_validation module

Cross Validation

Cross-validation support.

We assume that instances (i) are uniquely identified by an instance ID and (ii) they have id and score properties. We access them using the Instances class.

Authors Faegheh Hasibi, Krisztian Balog

class nordlys.core.ml.cross_validation.**CrossValidation**(*k*, *instances*, *callback_train*, *callback_test*)

Bases: `object`

Class attributes:

fold: dict of folds (1..k) with a dict {"training": [list of instance_ids]}, {"testing": [list of instance_ids]}

Parameters

- **k** – number of folds
- **instances** – Instances object
- **callback_train** – Callback function for training model
- **callback_test** – Callback function for applying model

create_folds(*group_by=None*)
Creates folds for the data set.

Parameters **group_by** – property to group by (instance_id by default)

get_folds(*filename=None*, *group_by=None*)
Loads folds from file or generates them if the file doesn't exist.

Parameters

- **filename** –
- **k** – number of folds

Returns

get_instances(*i*, *mode*, *property=None*)
Returns instances from the given fold i in [0..k-1].

Parameters

- **i** – fold number
- **mode** – training or testing

:return Instances object

load_folds(*filename*)
Loads previously created folds from (JSON) file.

run()
Runs cross-validation.

save_folds(*filename*)
Saves folds to (JSON) file.

nordlys.core.ml.cross_validation.**main**()

nordlys.core.ml.instance module

Instance

Instance class.

Features:

- This class supports different features for an instance.
- The features, together with the id of the instance, will be used in machine learning algorithms.
- All Features are stored in a dictionary, where keys are feature names (self.features).

Instance properties:

- Properties are additional side information of an instance (e.g. query_id, entity_id, . . .).
- properties are stored in a dictionary (self.properties).

This is the base instance class. Specific type of instances can inherit form class and add more properties to the base class.

Author Faegheh Hasibi

class nordlys.core.ml.instance.**Instance**(*id*, *features=None*, *target='0'*, *properties=None*)
Bases: object

Class attributes: ins_id: (string) features: a dictionary of feature names and values target: (string) target id or class properties: a dictionary of property names and values

add_feature (*feature*, *value*)
Adds a new feature to the features.

Parameters **feature** – (string), feature name

:param value

add_property (*property*, *value*)
Adds a new property to the properties.

Parameters **property** – (string), property name

:param value

features

classmethod from_json (*ins_id*, *fields*)
Reads an instance in JSON format and generates Instance.

Parameters

- **ins_id** – instance id
- **fields** – A dictionary of fields

:return (ml.Instance)

get_feature (*feature*)
Returns the value of a given feature.

:param feature :return value

get_property (*property*)
Returns the value of a given property.

:param property :return value

id

properties

to_json (*file_name=None*)
Converts instance to the JSON format.

Parameters **file_name** – (string)

:return JSON dump of the instance.

to_libsvm (*features*, *qid_prop=None*)
Converts instance to the Libsvm format. - RankLib format:

<target> qid:<qid> <feature>:<value> . . . # <info>

- Example: 3 qid:1 1:1 2:1 3:0 4:0.2 5:0 # 1A

NOTE: the property used for qid(qid_prop) should hold integers

Parameters

- **features** – the list of features that should be in the output
- **qid_prop** – property to be used as qid

:return str, instance in the rankLib format.

to_str(*feature_set=None*)
Converts instances to string.

Parameters **feature_set** – features to be included in the output format

Return (string) tab separated string ins_id target ftr_1 ftr_2 ... ftr_n properties

nordlys.core.ml.instance.**main**()

nordlys.core.ml.instances module

Instances

Instances used for Machine learning algorithms.

- Manages a set of Instance objects
- **Loads instance-data from JSON or TSV files**
  - When using TSV, instance properties, target, and features are loaded from separate files
- Generates a list of instances in JSON or RankLib format

Authors Faegheh Hasibi, Krisztian Balog

class nordlys.core.ml.instances.**Instances**(*instances=None*)
Bases: object

Class attributes: instances: Instance objects stored in a dictionary indexed by instance id

Parameters **instances** – instances in a list or dict - if list then list index is used as the instance ID - if dict then the key is used as the instance ID

add_features_from_tsv(*tsv_file*, *features*)

add_instance(*instance*)
Adds an Instance object to the list of instances.

Parameters **instance** – Instance object

add_properties_from_tsv(*tsv_file*, *properties*)

add_qids(*prop*)
Generates (integer) q_id-s (for libsvm) based on a given (non-integer) property. It assigns a unique integer value to each different value for that property.

Parameters **prop** – name of the property.

Returns

add_target_from_tsv(*tsv_file*)

append_instances(*ins_list*)
Appends the list of Instances objects.

Parameters **ins_list** – list of Instance objects

classmethod from_json(*json_file*)
Loads instances from a JSON file.

Parameters **json_file** – (string)

:return Instances object

get_all()
Returns list of all instances.

get_all_ids()
Returns list of all instance ids.

get_instance(*instance_id*)
Returns an instance by instance id.

Parameters **instance_id** – (string)

Returns Instance object

group_by_property(*property*)
Groups instances by a given property.

:param property :return a dictionary of instance ids {id:[ml.Instance, . . .], . . . }

to_json(*json_file=None*)
Converts all instances to JSON and writes it to the file

Parameters **json_file** – (string)

Returns JSON dump of all instances.

to_libsvm(*file_name=None*, *qid_prop=None*)
Converts all instances to the LibSVM format and writes them to the file. - Libsvm format:

<line> .=. <target> qid:<qid> <feature>:<value> . . . # <info> <target> .=. <float> <qid> .=. <positive integer> <feature> .=. <positive integer> <value> .=. <float> <info> .=. <string>

- Example: 3 qid:1 1:1 2:1 3:0 4:0.2 5:0 # 1A

NOTES:

- The property used for qid(qid_prop) should hold integers
- For pointwise algorithms, we use instance id for qid
- Lines in the RankLib input have to be sorted by increasing qid.

Parameters

- **file_name** – File to write libsvm format of instances.
- **qid_prop** – property to be used as qid. If none,

to_str(*file_name=None*)
Converts instances to string and write them to the given file. :param file_name :return: String format of instances

to_treceval(*file_name*, *qid_prop='qid'*, *docid_prop='en_id'*)
Generates a TREC style run file - If there is an entity ranked more than once for the same query, the one with higher score is kept.

Parameters

- **file_name** – File to write TREC file
- **qid_prop** – Name of instance property to be used as query ID (1st column)
- **docid_prop** – Name of instance property to be used as document ID (3rd column)

nordlys.core.ml.instances.**main**(*args*)

nordlys.core.ml.ml module

Machine leaning

The command-line application for general-purpose machine learning.

Usage

```
python -m nordlys.core.ml.ml <config_file>
```

Config parameters

- **training_set**: nordlys ML instance file format (MIFF)
- **test_set**: nordlys ML instance file format (MIFF); if provided then it's always used for testing. Can be left empty if cross-validation is used, in which case the remaining split is used for testing.
- **cross_validation:**
  - k: number of folds (default: 10); use -1 for leave-one-out
  - split_strategy: name of a property (normally query-id for IR problems). If set, the entities with the same value for that property are kept in the same split. if not set, entities are randomly distributed among splits.
  - splits_file: JSON file with splits (instance_ids); if the file is provided it is used, otherwise it's generated
  - create_splits: if True, creates the CV splits. Otherwise loads the splits from "split_file" parameter.
- **model**: ML model, currently supported values: rf, gbrt
- **category**: [regression | classification], default: "regression"
- **parameters: dict with parameters of the given ML model**
  - **If GBRT:**
    - alpha: learning rate, default: 0.1
    - tree: number of trees, default: 1000
    - depth: max depth of trees, default: 10% of number of features
  - **If RF:**
    - tree: number of trees, default: 1000
    - maxfeat: max features of trees, default: 10% of number of features
- **model_file**: the model is saved to this file
- **load_model**: if True, loads the model
- **feature_imp_file**: Feature importance is saved to this file
- **output_file**: where output is written; default output format: TSV with with instance_id and (estimated) target

Example config

```
{
    "model": "gbrt",
    "category": "regression",
        "parameters":{
                "alpha": 0.1,
                "tree": 10,
                "depth": 5
        },
        "training_set": "path/to/train.json",
        "test_set": "path/to/test.json",
        "model_file": "path/to/model.txt",
    "output_file": "path/to/output.json",
    "cross_validation":{
                "create_splits": true,
                "splits_file": "path/to/splits.json",
        "k": 5,
        "split_strategy": "q_id"
        }
}
```

Authors Faegheh Hasibi, Krisztian Balog

class nordlys.core.ml.ml.**ML**(*config*)

Bases: object

analyse_features(*model*, *feature_names*)

Ranks features based on their importance. Scikit uses Gini score to get feature importances.

Parameters

- **model** – trained model
- **feature_names** – list of feature names

apply_model(*instances*, *model*)

Applies model on a given set of instances.

Parameters

- **instances** – Instances object
- **model** – trained model

Returns Instances

gen_model(*num_features=None*)

Reads parameters and generates a model to be trained.

Parameters **num_features** – int, number of features

:return untrained ranker/classifier

output(*instances*)

Writes results to output file.

Parameters **instances** – Instances object

run()

train_model(*instances*)
Trains model on a given set of instances.

Parameters **instances** – Instances object

Returns the learned model

nordlys.core.ml.ml.**arg_parser**()

nordlys.core.ml.ml.**main**(*args*)

nordlys.core.retrieval package

Retrieval

The retrieval package provides basic indexing and scoring functionality based on Elasticsearch (v2.3). It can be used both for documents and for entities (as the latter are represented as fielded documents).

Indexing

Indexing can be done be done by directly reading the content of documents. The *toy_indexer* module provides a toy example.

When the content of documents is stored in MongoDB (e.g., for DBpedia entities), use the *indexer_mongo* module for indexing. For further details on how this module can be used, see indexer_dbpedia.

For indexing Dbpedia entities, we read the content of entiteis form MongoDB aFor DBpedia entities, we store them on MongoDB and .. todo:: Explain indexing (representing entities as fielded documents, mongo to elasticsearch)

Notes

- To speed up indexing, use *add_docs_bulk()*. The optimal number of documents to send in a single bulk depends on the size of documents; you need to figure it out experimentally.
- We strongly recommend using the default Elasticsearch similarity (currently BM25) for indexing. (Other similarity functions may be also used; in that case the similarity function can updated after indexing.)
- Our default setting is *not* to store term positions in the index (for efficiency considerations).

Retrieval

Retrieval is done in two stages:

- *First pass*: The top N documents are retrieved using Elastic's default search method
- *Second pass*: The (expensive) scoring of the top N documents is performed (implemented in the Nordlys)

Nordlys currently supports the following models for second pass retrieval:

- Language modelling (LM) [1]
- Mixture of Language Modesl (MLM) [2]
- Probabilistic Model for Semistructured Data (PRMS) [3]

Check out *scorer* module to get inspiration for implementing a new retrieval model.

Command line usage

See *nordlys.core.retrieval.retrieval*

Notes

- Always use a *ElasticCache* object (instead of *Elastic*) for getting stats from the index. This class stores index stats in the memory, which highly benefits efficiency.
- We recommend to create a new *ElasticCache* object for each query. This way, you will make effiecnt of your machine's memory.

---

[1] Jay M Ponte and W Bruce Croft. 1998. *A Language modeling approach to information retrieval*. In Proc. of SIGIR '98.

[2] Paul Ogilvie and Jamie Callan. 2003. *Combining document representations for known-item search*. Proc. of SIGIR '03.

[3] Jinyoung Kim, Xiaobing Xue, and W Bruce Croft. 2009. *A probabilistic retrieval model for semistructured data*. In Proc. of ECIR '09.

Submodules

nordlys.core.retrieval.elastic module

Elastic

Utility class for working with Elasticsearch. This class is to be instantiated for each index.

Indexing usage

To create an index, first you need to define field mappings and then build the index. The sample code for creating an index is provided at *nordlys.core.retrieval.toy_indexer*.

Retrieval usage

The following statistics can be obtained from this class:

- Number of documents: *Elastic.num_docs()*
- Number of fields: *Elastic.num_fields()*
- Document count: *Elastic.doc_count()*
- Collection length: *Elastic.coll_length()*
- Average length: *Elastic.avg_len()*
- Document length: *Elastic.doc_length()*
- Document frequency: *Elastic.doc_freq()*
- Collection frequency: *Elastic.coll_term_freq()*

- Term frequencies: *Elastic.term_freqs()*

Efficiency considerations

- For efficiency reasons, we do not store term positions during indexing. To store them, see the corresponding mapping functions *Elastic.analyzed_field()*, *Elastic.notanalyzed_searchable_field()*.
- Use *ElasticCache* for getting index statistics. This module caches the statistics into memory and boosts efficeicny.
- Mind that *ElasticCache* does not empty the cache!

Authors Faegheh Hasibi, Krisztian Balog

class nordlys.core.retrieval.elastic.**Elastic**(*index_name*)
Bases: object

ANALYZER_STOP = 'stop_en'

ANALYZER_STOP_STEM = 'english'

BM25 = 'BM25'

DOC_TYPE = 'doc'

FIELD_CATCHALL = 'catchall'

FIELD_ELASTIC_CATCHALL = '_all'

SIMILARITY = 'sim'

add_doc(*doc_id*, *contents*)
Adds a document with the specified contents to the index.

Parameters

- **doc_id** – document ID
- **contents** – content of document

add_docs_bulk(*docs*)
Adds a set of documents to the index in a bulk.

Parameters **docs** – dictionary {doc_id: doc}

analyze_query(*query*, *analyzer='stop_en'*)
Analyzes the query.

Parameters

- **query** – raw query
- **analyzer** – name of analyzer

static analyzed_field(*analyzer='stop_en'*)
Returns the mapping for analyzed fields.

For efficiency considerations, term positions are not stored. To store term positions, change `"term_vector": "with_positions_offsets"`

Parameters **analyzer** – name of the analyzer; valid options: [ANALYZER_STOP, ANALYZER_STOP_STEM]

avg_len(*field*)
Returns average length of a field in the collection.

coll_length(*field*)
Returns length of field in the collection.

coll_term_freq(*term*, *field*, *tv=None*)
Returns collection term frequency for the given field.

create_index(*mappings*, *model='BM25'*, *model_params=None*, *force=False*)
Creates index (if it doesn't exist).

Parameters

- **mappings** – field mappings
- **model** – name of elastic search similarity
- **model_params** – name of elastic search similarity
- **force** – forces index creation (overwrites if already exists)

delete_index()
Deletes an index.

doc_count(*field*)
Returns number of documents with at least one term for the given field.

doc_freq(*term*, *field*, *tv=None*)
Returns document frequency for the given term and field.

doc_length(*doc_id*, *field*)
Returns length of a field in a document.

get_doc(*doc_id*, *fields=None*, *source=True*)
Gets a document from the index based on its ID.

Parameters

- **doc_id** – document ID
- **fields** – list of fields to return (default: all)
- **source** – return document source as well (default: yes)

get_field_stats(*field*)
Returns stats of the given field.

get_fields()
Returns name of fields in the index.

get_mapping()
Returns mapping definition for the index.

get_settings()
Returns index settings.

static notanalyzed_field()
Returns the mapping for not-analyzed fields.

static notanalyzed_searchable_field()
Returns the mapping for not-analyzed fields.

num_docs()
Returns the number of documents in the index.

num_fields()
Returns number of fields in the index.

search(*query*, *field*, *num=100*, *fields_return=''*, *start=0*)
Searches in a given field using the similarity method configured in the index for that field.

Parameters

- **query** – query string
- **field** – field to search in
- **num** – number of hits to return (default: 100)
- **fields_return** – additional document fields to be returned
- **start** – starting offset (default: 0)

Returns dictionary of document IDs with scores

search_complex(*body*, *num=10*, *fields_return=''*, *start=0*)
Supports complex structured queries, which are sent as a `body` field in Elastic search. For detailed information on formulating structured queries, see the official instructions. Below is an example to search in two particular fields that each must contain a specific term.

Example

```
# [explanation of the query]
term_1 = "hello"
term_2 = "world"
body = {
    "query": {
        "bool": {
            "must": [
                {
                "match": {"title": term_1}
                },
                {
                "match_phrase": {"content": term_2}
                }
                ]
            }
        }
    }
```

Parameters

- **body** – query body
- **field** – field to search in
- **num** – number of hits to return (default: 100)
- **fields_return** – additional document fields to be returned
- **start** – starting offset (default: 0)

Returns dictionary of document IDs with scores

term_freq(*doc_id*, *field*, *term*)
Returns frequency of a term in a given document and field.

term_freqs(*doc_id*, *field*, *tv=None*)
Returns term frequencies of all terms for a given document and field.

update_similarity(*model='BM25'*, *params=None*)
Updates the similarity function "sim", which is fixed for all index fields.

The method and param should match elastic settings: https://www.elastic.co/guide/en/elasticsearch/reference/2.3/index-modules-similarity.html

Parameters

- **model** – name of the elastic model
- **params** – dictionary of params based on elastic

nordlys.core.retrieval.elastic_cache module

Elastic Cache

This is a cache for elastic index stats; a layer between an index and retrieval. The statistics (such as document and term frequencies) are first read from the index and stay in the memory for further usages.

Usage hints

- Only one instance of Elastic cache needs to be created.
- If running out of memory, you need to create a new object of ElasticCache.
- The class also caches termvectors. To further boost efficiency, you can load term vectors for multiple documents using *ElasticCache.multi_termvector()*.

Author Faegheh Hasibi

class nordlys.core.retrieval.elastic_cache.**ElasticCache**(*index_name*)
Bases: *nordlys.core.retrieval.elastic.Elastic*

avg_len(*field*)
Returns average length of a field in the collection.

coll_length(*field*)
Returns length of field in the collection.

coll_term_freq(*term*, *field*, *tv=None*)
Returns collection term frequency for the given field.

doc_count(*field*)
Returns number of documents with at least one term for the given field.

doc_freq(*term*, *field*, *tv=None*)
Returns document frequency for the given term and field.

doc_length(*doc_id*, *field*)
Returns length of a field in a document.

multi_termvector(*doc_ids*, *field*, *batch=50*)
Returns term vectors for a given document and field.

num_docs()
Returns the number of documents in the index.

num_fields()
Returns number of fields in the index.

term_freq(*doc_id*, *field*, *term*)
Returns frequency of a term in a given document and field.

term_freqs(*doc_id*, *field*, *tv=None*)
Returns term frequencies for a given document and field.

nordlys.core.retrieval.indexer_mongo module

Mongo Indexer

This class is a tool for creating an index from a Mongo collection.

To use this class, you need to implement `callback_get_doc_content()` function. See `indexer_fsdm` for an example usage of this class.

Author Faegheh Hasibi

class nordlys.core.retrieval.indexer_mongo.**IndexerMongo**(*index_name*, *mappings*, *collection*, *model='BM25'*)

Bases: `object`

build(*callback_get_doc_content*, *bulk_size=1000*)
Builds the DBpedia index from the mongo collection.

To speedup indexing, we index documents as a bulk. There is an optimum value for the bulk size; try to figure it out.

Parameters

- **callback_get_doc_content** – a function that get a documet from mongo and return the content for indexing
- **bulk_size** – Number of documents to be added to the index as a bulk

nordlys.core.retrieval.retrieval module

Retrieval

Console application for general-purpose retrieval.

Usage

```
python -m nordlys.services.er -c <config_file> -q <query>
```

If *-q <query>* is passed, it returns the results for the specified query and prints them in terminal.

Config parameters

- **index_name**: name of the index,
- **first_pass:**
  - **1st_num_docs**: number of documents in first-pass scoring (default: 100)

  - **field**: field used in first pass retrieval (default: Elastic.FIELD_CATCHALL)
  - **fields_return**: comma-separated list of fields to return for each hit (default: "")
- **num_docs**: number of documents to return (default: 100)
- **start**: starting offset for ranked documents (default:0)
- **model**: name of retrieval model; accepted values: [lm, mlm, prms] (default: lm)
- **field**: field name for LM (default: catchall)
- **fields: single field name for LM (default: catchall)** list of fields for PRMS (default: [catchall]) dictionary with fields and corresponding weights for MLM (default: {catchall: 1})
- **smoothing_method**: accepted values: [jm, dirichlet] (default: dirichlet)
- **smoothing_param**: value of lambda or mu; accepted values: [float or "avg_len"], (jm default: 0.1, dirichlet default: 2000)
- **query_file**: name of query file (JSON),
- **output_file**: name of output file,
- **run_id**: run id for TREC output

Example config

```
{"index_name": "dbpedia_2015_10",
  "first_pass": {
    "1st_num_docs": 1000
  },
  "model": "prms",
  "num_docs": 1000,
  "smoothing_method": "dirichlet",
  "smoothing_param": 2000,
  "fields": ["names", "categories", "attributes", "similar_entity_names", "related_
↪entity_names"],
  "query_file": "path/to/queries.json",
  "output_file": "path/to/output.txt",
  "run_id": "test"
}
```

Authors Krisztian Balog, Faegheh Hasibi

class nordlys.core.retrieval.retrieval.**Retrieval**(*config*)

Bases: object

FIELDED_MODELS = set(['mlm', 'prms'])

LM_MODELS = set(['lm', 'mlm', 'prms'])

batch_retrieval()

Scores queries in a batch and outputs results.

static check_config(*config*)

Checks config parameters and sets default values.

retrieve(*query*, *scorer=None*)

Scores documents for the given query.

trec_format(*results*, *query_id*, *max_rank=100*)
Outputs results in TREC format

nordlys.core.retrieval.retrieval.**arg_parser**()

nordlys.core.retrieval.retrieval.**get_config**()

nordlys.core.retrieval.retrieval.**main**(*args*)

nordlys.core.retrieval.retrieval_results module

Retrieval Results

Result list representation.

- for each hit it holds score and both internal and external doc_ids

Authors Faegheh Hasibi, Krisztian Balog

class nordlys.core.retrieval.retrieval_results.**RetrievalResults**(*scores={}*, *query=None*)
Bases: object

Class for storing retrieval scores for a given query.

append(*doc_id*, *score*)
Adds document to the result list

classmethod elastic_to_retrieval(*res*, *query=None*)
Converts elastic search results to retrieval results.

get_score(*doc_id*)
Returns the score of a document (or None if it's not in the list).

get_scores_sorted()
Returns all results sorted by score

num_docs()
Returns the number of documents in the result list.

query

write_trec_format(*query_id*, *run_id*, *out*, *max_rank=100*)
Outputs results in TREC format

nordlys.core.retrieval.scorer module

Scorer

Various retrieval models for scoring a individual document for a given query.

Authors Faegheh Hasibi, Krisztian Balog

class nordlys.core.retrieval.scorer.**Scorer**(*elastic*, *query*, *params*)
Bases: object

Base scorer class.

SCORER_DEBUG = 0

static get_scorer(*elastic*, *query*, *config*)
Returns Scorer object (Scorer factory).

Parameters

- **elastic** – Elastic object
- **query** – raw query (to be analyzed)
- **config** – dict with models parameters

class nordlys.core.retrieval.scorer.**ScorerLM**(*elastic*, *query*, *params*)
Bases: *nordlys.core.retrieval.scorer.Scorer*

Language Model (LM) scorer.

DIRICHLET = 'dirichlet'

JM = 'jm'

static get_dirichlet_prob(*tf_t_d*, *len_d*, *tf_t_C*, *len_C*, *mu*)
Computes Dirichlet-smoothed probability. P(t|theta_d) = [tf(t, d) + mu P(t|C)] / [|d| + mu]

Parameters

- **tf_t_d** – tf(t,d)
- **len_d** – |d|
- **tf_t_C** – tf(t,C)
- **len_C** – |C| = sum_{d in C} |d|
- **mu** – mu

Returns Dirichlet-smoothed probability

static get_jm_prob(*tf_t_d*, *len_d*, *tf_t_C*, *len_C*, *lambd*)
Computes JM-smoothed probability. p(t|theta_d) = [(1-lambda) tf(t, d)/|d|] + [lambda tf(t, C)/|C|]

Parameters

- **tf_t_d** – tf(t,d)
- **len_d** – |d|
- **tf_t_C** – tf(t,C)
- **len_C** – |C| = sum_{d in C} |d|
- **lambd** – lambda

Returns JM-smoothed probability

get_lm_term_prob(*doc_id*, *field*, *t*, *tf_t_d_f=None*, *tf_t_C_f=None*)
Returns term probability for a document and field.

Parameters

- **doc_id** – document ID
- **field** – field name
- **t** – term

Returns P(t|d_f)

get_lm_term_probs(*doc_id*, *field*)
Returns probability of all query terms for a document and field; i.e. p(t|theta_d)

Parameters

- **doc_id** – document ID
- **field** – field name

Returns dictionary of terms with their probabilities

score_doc(*doc_id*)
Scores the given document using LM. p(q|theta_d) = sum log(p(t|theta_d))

Parameters **doc_id** – document id

Returns LM score

class nordlys.core.retrieval.scorer.**ScorerMLM**(*elastic*, *query*, *params*)
Bases: *nordlys.core.retrieval.scorer.ScorerLM*

Mixture of Language Model (MLM) scorer.

Implemented based on: Ogilvie, Callan. Combining document representations for known-item search. SIGIR 2003.

get_mlm_term_prob(*doc_id*, *t*)
Returns MLM probability for the given term and field-weights. p(t|theta_d) = sum(mu_f * p(t|theta_d_f))

Parameters

- **lucene_doc_id** – internal Lucene document ID
- **t** – term

Returns P(t|theta_d)

get_mlm_term_probs(*doc_id*)
Returns probability of all query terms for a document; i.e. p(t|theta_d)

Parameters **doc_id** – internal Lucene document ID

Returns dictionary of terms with their probabilities

score_doc(*doc_id*)
Scores the given document using MLM model. p(q|theta_d) = sum log(p(t|theta_d))

Parameters **doc_id** – document ID

Returns MLM score of document and query

class nordlys.core.retrieval.scorer.**ScorerPRMS**(*elastic*, *query*, *params*)
Bases: *nordlys.core.retrieval.scorer.ScorerLM*

PRMS scorer.

get_mapping_prob(*t*, *coll_termfreq_fields=None*)

Computes PRMS field mapping probability. p(f|t) = P(t|f)P(f) / sum_f'(P(t|C_{f'_c})P(f'))

Parameters

- **t** – str
- **coll_termfreq_fields** – {field: freq, ... }

Returns a dictionary {field: prms_prob, ... }

get_mapping_probs()
Gets (cached) mapping probabilities for all query terms.

get_total_field_freq()
Returns total occurrences of all fields

score_doc(*doc_id*)
Scores the given document using PRMS model.

Parameters

- **doc_id** – document id
- **lucene_doc_id** – internal Lucene document ID

Returns float, PRMS score of document and query

nordlys.core.retrieval.toy_indexer module

Toy Indexer

Toy indexing example for testing purposes.

Authors Krisztian Balog, Faegheh Hasibi

nordlys.core.retrieval.toy_indexer.**main**()

nordlys.core.storage package

This is the (**intro** of the) storage package doc. It's in the `__init__.py` of the package, as an example for the other packages, and IT SHOULD BE cleaned up without all of this ;).

No idea what text it could contain :)

It might contain, e.g., items in this way

- This
- That

and also code snippets like usual, using as usual these double colons and the indented code:

```
mongo = Mongo(host, db, collection)
```

And that's it.

(In the following I included all the up-to-now available modules, sectioned by no criteria in particular.)

Subpackages

nordlys.core.storage.parser package

Submodules

nordlys.core.storage.parser.nt_parser module

NTriples Parser

NTriples parser with URI prefixing

Author Krisztian Balog

class nordlys.core.storage.parser.nt_parser.**NTParser**
Bases: object

NTriples parser class

parse_file(*filename*, *triplehandler*)
Parses file and calls callback function with the parsed triple

class nordlys.core.storage.parser.nt_parser.**Triple**(*prefix=None*)
Bases: *object*

Representation of a Triple to be used by the rdflib NTriplesParser.

object()

object_prefixed()

predicate()

predicate_prefixed()

subject()

subject_prefixed()

triple(*s*, *p*, *o*)
Assign current triple object

Parameters

- **s** – subject
- **p** – predicate
- **o** – object

class nordlys.core.storage.parser.nt_parser.**TripleHandler**
Bases: object

This is an abstract class

triple_parsed(*triple*)
This method is called each time a triple is parsed, with the triple as parameter.

class nordlys.core.storage.parser.nt_parser.**TripleHandlerPrinter**
Bases: *nordlys.core.storage.parser.nt_parser.TripleHandler*

Example triple handler that only prints whatever it received.

triple_parsed(*triple*)
This method is called each time a triple is parsed, with the triple as parameter.

nordlys.core.storage.parser.nt_parser.**main**(*argv*)

nordlys.core.storage.parser.uri_prefix module

URI Prefixing

URI prefixing.

Author Krisztian Balog

class nordlys.core.storage.parser.uri_prefix.**URIPrefix**(*prefix_file='/home/docs/checkouts/readthedocs.org/u*
Bases: object

get_prefixed(*uri*, *angle_brackets=True*)

nordlys.core.storage.parser.uri_prefix.**convert_txt_to_json**(*txt_file*, *json_file='/home/docs/checkouts/readthedocs.c*

Convert prefixes txt file to json.

This has to be done only once. And only in case there is no .json file, or any changes done in .txt.

Submodules

nordlys.core.storage.mongo module

Mongo

Tools for working with MongoDB.

Authors Krisztian Balog, Faegheh Hasibi

class nordlys.core.storage.mongo.**Mongo**(*host*, *db*, *collection*)
Bases: object

Manages the MongoDB connection and operations.

ID_FIELD = '_id'

add(*doc_id*, *contents*)
Adds a document or replaces the contents of an entire document.

append_dict(*doc_id*, *field*, *dictkey*, *value*)
Appends the value to a given field that stores a dict. If the dictkey is already in use, the value stored there will be overwritten.

Parameters

- **doc_id** – document id
- **field** – field
- **dictkey** – key in the dictionary
- **value** – value to be increased by

append_list(*doc_id*, *field*, *value*)
Appends the value to a given field that stores a list. If the field does not exist yet, it will be created. The value should be a list.

Parameters

- **doc_id** – document id
- **field** – field
- **value** – list, a value to be appended to the current list

append_set(*doc_id*, *field*, *value*)
Adds a list of values to a set. If the field does not exist yet, it will be created. The value should be a list.

Parameters

- **doc_id** – document id

- **field** – field
- **value** – list, a value to be appended to the current list

drop()
Deletes the contents of the given collection (including indices).

find_all(*no_timeout=False*)
Returns a Cursor instance that allows us to iterate over all documents.

find_by_id(*doc_id*)
Returns unescaped document content for a given document id.

get_num_docs()
Returns total number of documents in the mongo collection.

inc(*doc_id*, *field*, *value*)
Increments the value of a specified field.

inc_in_dict(*doc_id*, *field*, *dictkey*, *value=1*)
Increments a value that is inside a dict.

Parameters

- **doc_id** – document id
- **field** – field
- **dictkey** – key in the dictionary
- **value** – value to be increased by

static print_doc(*doc*)

set(*doc_id*, *field*, *value*)
Sets the value of a given document field (overwrites previously stored content).

static unescape(*s*)
Unescapes string.

static unescape_doc(*mdoc*)
Unescapes document content.

nordlys.core.storage.mongo.**main**()

nordlys.core.storage.nt2mongo module

nordlys.core.utils package

@author: Krisztian Balog

Submodules

nordlys.core.utils.entity_utils module

nordlys.core.utils.file_utils module

File Utils

Utility methods for file handling.

Authors Krisztian Balog, Faegheh Hasibi

class nordlys.core.utils.file_utils.**FileUtils**
Bases: object

static dump_tsv(*file_name*, *data*, *header=None*, *append=False*)
Dumps the data in tsv format.

Parameters

- **file_name** – name of file
- **data** – list of list
- **header** – list of headers
- **append** – if True, appends the data to the existing file

static load_config(*config*)
Loads config file/dictionary.

Parameters **config** – json file or a dictionary

Returns config dictionary

static open_file_by_type(*file_name*, *mode='r'*)
Opens file (gz/text) and returns the handler.

Parameters **file_name** – NTriples file

Returns handler to the file

static read_file_as_list(*filename*)
Reads in non-empty lines from a textfile (which may be gzipped/bz2ed) and returns it as a list.

Parameters **filename** –

nordlys.core.utils.file_utils.**main**()

nordlys.core.utils.logging_utils module

Logging Utils

Utility methods for logging.

Author Heng Ding

class nordlys.core.utils.logging_utils.**PrintHandler**(*logging_level*)
Bases: object

Handler for elastic prints

class nordlys.core.utils.logging_utils.**RequestHandler**(*logging_path*)
Bases: object

Handler for elastic request

nordlys.logic package

Services

All modules in this package serving for services layer.

Subpackages

nordlys.logic.el package

Entity linking

This package is the implementation of entity linking.

Submodules

nordlys.logic.el.cmns module

Commonness Entity Linking Approach

Class for commonness entity linking approach

Author Faegheh Hasibi

class nordlys.logic.el.cmns.**Cmns**(*query*, *entity*, *threshold=None*, *cmns_th=0.1*)
Bases: object

disambiguate()
Selects only one entity per mention.

Return [{"mention" xx, "entity": yy, "score": zz}, . . .] #dictionary {mention: (en_id, score), ..}

link()
Links the query to the entity.

dictionary {mention: (en_id, score), ..}

rank_ens()
Detects mention and rank entities for each mention

nordlys.logic.el.cmns.**main**(*args*)

nordlys.logic.el.el_utils module

EL Utils

Utility methods for entity linking.

Author: Faegheh Hasibi

nordlys.logic.el.el_utils.**is_name_entity**(*en_id*)
Returns true if the entity is considered as proper name entity.

nordlys.logic.el.el_utils.**load_kb_snapshot**(*kb_file*)
Loads DBpedia Snapshot of proper name entities (used for entity linking).

nordlys.logic.el.el_utils.**to_elq_eval**(*annotations*, *output_file*)
Write entity annotations to ELQ evaluation format.

Parameters **linked_ens** – {qid:[{"mention":xx, "entity": yy, "score":zz}, ..], ..}

nordlys.logic.el.greedy module

Generative model for interpretation set finding

@author: Faegheh Hasibi

class nordlys.logic.el.greedy.**Greedy**(*score_th*)
Bases: object

create_interpretations(*query_inss*)
Groups CER instances as interpretation sets.

Return list of interpretations, where each interpretation is a dictionary {mention (en_id, score), ..}

disambiguate(*inss*)
Takes instances and generates set of entity linking interpretations.

Parameters **inss** – Instances object

Returns sets of interpretations [{mention: (en_id, score), ..}, . . .]

is_overlapping(*mentions*)
Checks whether the strings of a set overlapping or not. i.e. if there exists a term that appears twice in the whole set.

E.g. {"the", "music man"} is not overlapping {"the", "the man", "music"} is overlapping.

NOTE: If a query is "yxxz" the mentions {"yx", "xz"} and {"yx", "x"} are overlapping.

Parameters **mentions** – A list of strings

:return True/False

prune_by_score(*query_inss*)
prunes based on a static threshold of ranking score.

prune_containment_mentions(*query_inss*)
Deletes containment mentions, if they have lower score.

nordlys.logic.el.ltr module

LTR Entity Linking Approach

Class for Learning-to-Rank entity linking approach

Author Faegheh Hasibi

class nordlys.logic.el.ltr.**LTR**(*query*, *entity*, *elastic*, *fcache*, *model=None*, *threshold=None*, *cmns_th=0.1*)
Bases: object

disambiguate(*inss*)
Performs disambiguation

static gen_train_set(*gt*, *query_file*, *train_set*)
Trains LTR model for entity linking.

get_candidate_inss()
Detects mentions and their candidate entities (with their commoness scores) and generates instances

Returns Instances object

get_features(*ins*, *cand_ens=None*)
Generates the features set for each instance.

Parameters

- **ins** – instance object
- **cand_ens** – dictionary of candidate entities {en_id: cmns, . . . }

Returns dictionary of features {ftr_name: value, . . . }

link()
Links the query to the entity.

Returns dictionary [{"mention": xx, "entity": yy, "score": zz}, . . .]

static load_yerd(*gt_file*)
Reads the Y-ERD collection and returns a dictionary.

Parameters **gt_file** – Path to the Y-ERD collection

Returns dictionary {(qid, query, en_id, mention) . . . }

rank_ens()
Ranks instances according to the learned LTR model

Parameters **n** – length of n-gram

Returns dictionary {(dbp_uri, fb_id):commonness, ..}

static train(*config*)

nordlys.logic.elr package

ELR

This package is the implementation of ELR-based models.

Submodules

nordlys.logic.elr.field_mapping module

Field Mapping for ELR

Computes PRMS field mapping probabilities.

Author Faegheh Hasibi

class nordlys.logic.elr.field_mapping.**FieldMapping**(*elastic_uri*, *n*)
Bases: object

DEBUG = 0

MAPPING_DEBUG = 0

map(*en_id*)
Gets PRMS mapping probability for a clique type

Return Dictionary {field weight, ..}

nordlys.logic.elr.field_mapping.**arg_parser**()

nordlys.logic.elr.field_mapping.**load_entities**(*annot_file*, *th=0.1*)

nordlys.logic.elr.field_mapping.**main**(*args*)

nordlys.logic.elr.scorer_elr module

nordlys.logic.elr.top_fields module

Top Fields

This class returns top fields based on document frequency

Author Faegheh Hasibi

class nordlys.logic.elr.top_fields.**TopFields**(*elastic*)
Bases: object

DEBUG = 0

fields

get_top_term(*en*, *n*)
Returns top-n fields with highest document frequency for the given entity ID.

nordlys.logic.entity package

Entity

This is the entity package.

Submodules

nordlys.logic.entity.entity module

Entity

Provides access to entity catalogs (DBpedia and surface forms).

Author Faegheh Hasibi

class nordlys.logic.entity.entity.**Entity**
Bases: object

dbp_to_fb(*dbp_id*)
Converts DBpedia id to Freebase; it returns list of Freebase IDs.

fb_to_dbp(*fb_id*)
Converts Freebase id to DBpedia; it returns list of DBpedia IDs.

lookup_en(*entity_id*)
Looks up an entity by its identifier.

Parameters **entity_id** – entity identifier ("<dbpedia:Audi_A4>")

:return A dictionary with the entity document or None.

lookup_name_dbpedia(*name*)
Looks up a name in a surface form dictionary and returns all candidate entities.

lookup_name_facc(*name*)
Looks up a name in a surface form dictionary and returns all candidate entities.

nordlys.logic.er package

Entity retrieval

This is the entity retrieval package.

Submodules

nordlys.logic.er.entity_retrieval module

nordlys.logic.er.field_mapping module

Field Mapping for ER

Computes PRMS field mapping probabilities.

Author Faegheh Hasibi

class nordlys.logic.er.field_mapping.**FieldMapping**(*elastic_uri*, *n*)
Bases: object

DEBUG = 0

MAPPING_DEBUG = 0

map(*en_id*)
Gets PRMS mapping probability for a clique type

Return Dictionary {field weight, ..}

nordlys.logic.er.field_mapping.**arg_parser**()

nordlys.logic.er.field_mapping.**load_entities**(*annot_file*, *th=0.1*)

nordlys.logic.er.field_mapping.**main**(*args*)

nordlys.logic.er.scorer_elr module

nordlys.logic.er.top_fields module

Top Fields

This class returns top fields based on document frequency

Author Faegheh Hasibi

class nordlys.logic.er.top_fields.**TopFields**(*elastic*)
Bases: object

DEBUG = 0

fields

get_top_term(*en*, *n*)
Returns top-n fields with highest document frequency for the given entity ID.

nordlys.logic.features package

Features

This is the features package.

Submodules

nordlys.logic.features.feature_cache module

Feature

Implements a generic feature class.

Authors: Faegheh Hasibi

class nordlys.logic.features.feature_cache.**FeatureCache**
Bases: object

get_feature_val(*feature_name*, *key*, *callback_func*, **args*)
Checks the cache and computes the feature if it does not exists

set_feature_val(*feature_name*, *key*, *value*)
Adds a feature and its value to the cache.

Parameters

- **feature_name** – Name of the feature
- **key** – the name of what feature is computed for (e.g., a mention, entity)
- **value** – feature value

nordlys.logic.features.ftr_entity module

FTR Entity

Implements features related to an entity.

Author Faegheh Hasibi

class nordlys.logic.features.ftr_entity.**FtrEntity**(*en_id*, *entity*)
Bases: object

outlinks()
Number of entity out-links

redirects()
Number of redirect pages linking to the entity

nordlys.logic.features.ftr_entity_mention module

FTR Entity Mention

Implements features related to an entity-mention pair.

Author Faegheh Hasibi

class nordlys.logic.features.ftr_entity_mention.**FtrEntityMention**(*en_id*, *mention*, *entity*)
Bases: object

commonness()
Computes probability of entity e being linked by mention: link (e,m)/link(m) Returns zero if link(m) = 0

mct()
Returns True if mention contains the title of entity

pos1()
Returns position of the occurrence of mention in the short abstract.

tcm()
Returns True if title of entity contains mention

tem()
Returns True if title of entity equals mention.

nordlys.logic.features.ftr_entity_similarity module

FTR Entity Similarity

Implements features capturing the similarity between entity and a query.

Author Faegheh Hasibi

class nordlys.logic.features.ftr_entity_similarity.**FtrEntitySimilarity**(*query*, *en_id*, *elastic*)
Bases: object

DEBUG = 0

context_sim(*mention*, *field='catchall'*)

LM score of entity to the context of query (context means query - mention)

E.g. given the query "uss yorktown charleston" and mention "uss", query context is " yorktown charleston"

Parameters

- **mention** – string
- **field** – field name

:return context similarity score

lm_score(*field='catchall'*)

Query length normalized LM score between entity field and query

Parameters **field** – field name

:return MLM score

mlm_score(*field_weights*)

Query length normalized MLM similarity between the entity and query

Parameters **field_weights** – dictionary {field: weight, ...}

:return MLM score

nllr(*query*, *field_weights*)

Computes Normalized query likelihood (NLLR): NLLR(q,d) = sum_{t in q} P(t|q) log P(t| heta_d) - sum_{t in q} p(t|q) log P(t|C) where:

P(t|q) = n(t,q)/|q| P(t|C) = sum_{f} mu_f * P(t|C_f) P(t| heta_d) = smoothed LM/MLM score

Parameters

- **query** – query
- **field_weights** – dictionary {field: weight, ...}

Returns NLLR score

nordlys.logic.features.ftr_lexical module

nordlys.logic.features.ftr_mention module

FTR Mention

Implements mention feature.

Author Faegheh Hasibi

class nordlys.logic.features.ftr_mention.**FtrMention**(*mention*, *entity=None*, *cand_ens=None*)

Bases: object

len_ratio(*q*)

Computes mention to query length.

matches()
Number of entities whose surface form equals the mention. Uses both DBpedia and Freebase name variants.

mention_len()
Number of terms in the mention

nordlys.logic.features.word2vec module

Word2vec

Implements functionalities over the 300-dim GoogleNews word2vec semantic representations of words.

Author Dario Garigliotti

class nordlys.logic.features.word2vec.**Word2Vec**(*mongo*)
Bases: object

get_centroid_vector(*s*)
Returns the normalized sum of the word2vec vectors corresponding to the terms in s.

Parameters **s** (*str*) – a phrase.

Returns Centroid vector of the terms in s.

get_vector(*word*)
Gets the w2v vector corresponding to the word, or a zero-valued vector if not present.

Parameters **word** (*str*) – a word.

Returns

nordlys.logic.features.word2vec.**arg_parser**()

nordlys.logic.features.word2vec.**main**(*args*)

nordlys.logic.fusion package

Submodules

nordlys.logic.fusion.fusion_scorer module

Fusion Scorer

Abstract class for fusion-based scoring.

Authors Shuo Zhang, Krisztian Balog, Dario Garigliotti

class nordlys.logic.fusion.fusion_scorer.**FusionScorer**(*index_name*, *association_file=None*, *run_id='fusion'*)
Bases: object

Parameters

- **index_name** – name of index
- **association_file** – association file

ASSOC_MODE_BINARY = 1

ASSOC_MODE_UNIFORM = 2

Abstract class for any fusion-based method.

load_associations()
Loads the document-object associations.

load_queries(*query_file*)
Loads the query file :return: query dictionary {queryID:query([term1,term2,...])}

score_queries(*queries*, *output_file*)
Scores all queries and optionally dumps results into an output file.

score_query(*query*, *assoc_fun=None*)

nordlys.logic.fusion.late_fusion_scorer module

Late Fusion Scorer

Class for late fusion scorer (i.e., document-centric model).

Authors Shuo Zhang, Krisztian Balog, Dario Garigliotti

class nordlys.logic.fusion.late_fusion_scorer.**LateFusionScorer**(*index_name*, *retr_model*, *retr_params*, *num_docs=None*, *field='content'*, *run_id='fusion'*, *num_objs=100*, *assoc_mode=1*, *assoc_file=None*)

Bases: *nordlys.logic.fusion.fusion_scorer.FusionScorer*

Parameters

- **index_name** – name of index
- **assoc_file** – document-object association file
- **assoc_mode** – document-object weight mode, uniform or binary
- **retr_model** – the retrieval model; valid values: "lm", "bm25"
- **retr_params** – config including smoothing method and parameter
- **num_objs** – the number of ranked objects for a query
- **assoc_mode** – the fusion weights, which could be binary or uniform
- **assoc_file** – object-doc association file

score_query(*query*, *assoc_fun=None*)
Scores a given query.

Parameters **query** – query string.

Returns a RetrievalResults instance.

Func assoc_fun function to return a list of docs for an obeject

nordlys.logic.query package

Query

This is the query package.

Submodules

nordlys.logic.query.mention module

Mention

Class for entity mentions (used for entity linking)

- Generates all candidate entities for a mention
- Computes commonness for a mention-entity pairs

class nordlys.logic.query.mention.**Mention**(*mention*, *entity*, *cmns_th=None*)
Bases: object

get_cand_ens()
Returns all candidate entities for the mention

Returns {en:cmn_score}

nordlys.logic.query.mention.**main**(*args*)

nordlys.logic.query.query module

Query

Class for representing a query.

TODO: add preprocessing using

Author Faegheh Hasibi

class nordlys.logic.query.query.**Query**(*query*, *qid=''*)
Bases: object

get_ngrams()
Finds all n-grams of the query.

Returns list of n-grams

get_terms()
Gets query terms.

Returns list of query terms

qid

query

raw_query

nordlys.logic.tti package

Submodules

nordlys.logic.tti.type_centric module

Type centric method for TTI.

@author:

class nordlys.logic.tti.type_centric.**TypeCentric**(*query*, *retrieval_config*)
Bases: object

nordlys.services package

Services

All modules in this package can be used from the command line or from a RESTful API.

Submodules

nordlys.services.api module

Nordlys API

This is the main console application for the Nordlys API.

Authors Krisztian Balog, Faegheh Hasibi, Shuo Zhang

nordlys.services.api.**after_request**(*response*)

nordlys.services.api.**catalog_dbp2fb**(*dbp_id*)

nordlys.services.api.**catalog_fb2dbp**(*fb_id*)

nordlys.services.api.**catalog_lookup_id**(*entity_id*)

nordlys.services.api.**catalog_lookup_sf_dbpedia**(*sf*)

nordlys.services.api.**catalog_lookup_sf_facc**(*sf*)

nordlys.services.api.**entity_linking**()

nordlys.services.api.**entity_types**()

nordlys.services.api.**error**(*str*)
@todo complete error handling

Parameters **str** –

Returns

nordlys.services.api.**exceptions**(*e*)

nordlys.services.api.**index**()

nordlys.services.api.**retrieval**()

nordlys.services.ec module

Entity catalog

Command line end point for entity catalog

Usage

python -m nordlys.services.ec -o <operation> -i <input>

Examples

- python -m nordlys.services.ec -o lookup_id -i <dbpedia:Audi_A4>
- python -m nordlys.services.ec -o "lookup_sf_dbpedia" -i "audi a4"
- python -m nordlys.services.ec -o "lookup_sf_facc" -i "audi a4"
- python -m nordlys.services.ec -o "dbpedia2freebase" -i "<dbpedia:Audi_A4>"
- python -m nordlys.services.ec -o "freebase2dbpedia" -i "<fb:m.030qmx>"

Author Faegheh Hasibi

nordlys.services.ec.**arg_parser**()

nordlys.services.ec.**main**(*args*)

nordlys.services.el module

Entity Linking

The command-line application for entity linking

Usage

```
python -m nordlys.services.el -c <config_file> -q <query>
```

If *-q <query>* is passed, it returns the results for the specified query and prints them in terminal.

Config parameters

- **method: name of the method**
  - **cmns** The baseline method that uses the overall popularity of entities as link targets
  - **ltr** The learning-to-rank model
- **threshold**: Entity linking threshold; varies depending on the method *(default: 0.1)*
- **step**: The step of entity linking process: [linking|ranking|disambiguation], *(default: linking)*
- **kb_snapshot**: File containing the KB snapshot of proper named entities; required for LTR, and optional for CMNS

- **query_file**: name of query file (JSON)
- **output_file**: name of output file

Parameters of LTR method:

- **model_file**: The trained model file; *(default:"data/el/model.txt")*
- **ground_truth**: The ground truth file; *(optional)*
- **gen_training_set**: If True, generates the training set from the groundtruth and query files; *(default: False)*
- **gen_model**: If True, trains the model from the training set; *(default: False)*
- The other parameters are similar to the nordlys.core.ml.ml settings

Example config

```
{
  "method": "cmns",
  "threshold": 0.1,
  "query_file": "path/to/queries.json"
  "output_file": "path/to/output.json"
}
```

Author Faegheh Hasibi

class nordlys.services.el.**EL**(*config*, *entity*, *elastic=None*, *fcache=None*)

Bases: object

batch_linking()

Scores queries in a batch and outputs results.

link(*query*, *qid=''*)

Performs entity linking for the query.

Parameters **query** – query string

Returns annotated query

nordlys.services.el.**arg_parser**()

nordlys.services.el.**main**(*args*)

nordlys.services.er module

Entity Retrieval

Command-line application for entity retrieval.

Usage

```
python -m nordlys.services.er -c <config_file> -q <query>
```

If *-q <query>* is passed, it returns the results for the specified query and prints them in terminal.

Config parameters

- **index_name**: name of the index,
- **first_pass:**
  - **num_docs**: number of documents in first-pass scoring (default: 100)
  - **field**: field used in first pass retrieval (default: Elastic.FIELD_CATCHALL)
  - **fields_return**: comma-separated list of fields to return for each hit (default: "")
- **num_docs**: number of documents to return (default: 100)
- **start**: starting offset for ranked documents (default:0)
- **model**: name of retrieval model; accepted values: [lm, mlm, prms] (default: lm)
- **field**: field name for LM (default: catchall)
- **fields**: list of fields for PRMS (default: [catchall])
- **field_weights**: dictionary with fields and corresponding weights for MLM (default: {catchall: 1})
- **smoothing_method**: accepted values: [jm, dirichlet] (default: dirichlet)
- **smoothing_param**: value of lambda or mu; accepted values: [float or "avg_len"], (jm default: 0.1, dirichlet default: 2000)
- **query_file**: name of query file (JSON),
- **output_file**: name of output file,
- **run_id**: run id for TREC output

Example config

```
{"index_name": "dbpedia_2015_10",
  "first_pass": {
    "num_docs": 1000
  },
  "model": "prms",
  "num_docs": 1000,
  "smoothing_method": "dirichlet",
  "smoothing_param": 2000,
  "fields": ["names", "categories", "attributes", "similar_entity_names", "related_
↪entity_names"],
  "query_file": "path/to/queries.json",
  "output_file": "path/to/output.txt",
  "run_id": "test"
}
```

Author Faegheh Hasibi

class nordlys.services.er.**ER**(*config*, *elastic=None*)

Bases: object

batch_retrieval()

Performs batch retrieval for a set of queries

retrieve(*query*)
Retrieves entities for a query

nordlys.services.er.**arg_parser**()

nordlys.services.er.**main**(*args*)

nordlys.services.tti module

Target Type Identification

The command-line application for target type identification.

Usage

```
python -m nordlys.services.tti <config_file> -q <query>
```

If *-q <query>* is passed, it returns the results for the specified query and prints them in terminal.

Config parameters

- **method**: name of TTI method; accepted values: ["tc", "ec", "ltr"]
- **num_docs**: number of documents to return
- **start**: starting offset for ranked documents
- **model**: retrieval model, if method is "tc" or "ec"; accepted values: ["lm", "bm25"]
- **ec_cutoff**: if method is "ec", rank cut-off of top-*K* entities for EC TTI
- **field**: field name, if method is "tc" or "ec"
- **smoothing_method**: accepted values: ["jm", "dirichlet"]
- **smoothing_param**: value of lambda or mu; accepted values: [float or "avg_len"]
- **query_file**: path to query file (JSON)
- **output_file**: path to output file (JSON)
- **trec_output_file**: path to output file (trec_eval-formatted)

Example config

```
{ "method": "ec",
  "num_docs": 10,
  "model": "lm",
  "first_pass": {
      "num_docs": 50
  },
  "smoothing_method": "dirichlet",
  "smoothing_param": 2000,
  "ec_cutoff": 20,
      "query_file": "path/to/queries.json",
```

(continues on next page)

(continued from previous page)

```
    "output_file": "path/to/output.txt",
}
```

Author Dario Garigliotti

class nordlys.services.tti.**TTI**(*config*)

Bases: object

batch_identification()

Annotates, in a batch, queries with identified target types, and outputs results.

identify(*query*)

Performs target type identification for the query.

Parameters **query** (*str*) – query string

Returns annotated query

nordlys.services.tti.**arg_parser**()

nordlys.services.tti.**main**(*args*)

2.7.2 Submodules

nordlys.config module

config

Global nordlys config.

Author Krisztian Balog

Author Faegheh Hasibi

nordlys.config.**load_nordlys_config**(*file_name*)

Loads nordlys config file. If local file is provided, global one is ignored.

2.8 Contact

Nordlys is written by:

- Faegheh Hasibi
- Krisztian Balog
- Dario Garigliotti
- Shuo Zhang

For any queries contact <faegheh.hasibi@ntnu.no> or <krisztian.balog@uis.no>.

Drop us some words! We are looking forward to hear your feedback!

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

n

Index

A

B

C

D

I

J

L

M

N

T

U

W