
entity-fishing documentation

Release 0.0.5

Patrice Lopez

Jul 07, 2022

Contents

1	Overview	1
1.1	Motivation	1
1.2	Tasks	1
1.3	Summary	4
1.4	How to cite	5
1.5	License and contact	5
2	Install, build, run, and monitor	7
2.1	Install, build, and run	7
2.2	Metrics and monitoring	9
2.3	Creating a new Knowledge Base version	9
3	<i>entity-fishing</i> Console	11
4	<i>entity-fishing</i> REST API	15
4.1	<i>entity-fishing</i> query processing	15
4.2	Knowledge base concept retrieval	28
4.3	Term Lookup	30
4.4	Language identification	31
4.5	Sentence segmentation	32
4.6	Customisation API	33
5	Evaluation	39
5.1	Datasets for long texts	39
5.2	Evaluation commands	40
5.3	Generation of pre-annotated training/evaluation data	40
5.4	References	40
6	Train and evaluate	43
6.1	Training with Wikipedia	43
6.2	Evaluation with Wikipedia	44
6.3	Training with an annotated corpus	44
6.4	Creating entity embeddings	44
7	License and contact	47

1.1 Motivation

One of the backbone of the activities of scientists regarding technical and scientific information at large is the identification and resolution of specialist entities. This could be the identification of scientific terms, of nomenclature-based expressions such as chemical formula, of quantity expressions, etc. It is considered that between 30 to 80% of the content of a technical or scientific document is written in specialist language (Ahmad, 1996). Researchers in Digital Humanities and in Social Sciences are often first of all interested in the identification and resolution of so-called named entities, e.g. person names, places, events, dates, organisation, etc. Entities can be known in advance and present in generalist or specialized knowledge bases. They can also be created based on open nomenclatures and vocabularies and impossible to enumerate in advance.

The *entity-fishing* services try to automate this recognition and disambiguation task in a generic manner, avoiding as much as possible restrictions of domains, limitations to certain classes of entities or to particular usages.

1.2 Tasks

entity-fishing performs the following tasks:

- entity recognition and disambiguation against Wikidata in a raw text, partially-annotated text segment,

CIGARETTE SMOKE CS-induced AIRWAY EPITHELIAL SENESENCE has been implicated in the PATHOGENESIS of CHRONIC OBSTRUCTIVE PULMONARY DISEASE COPD although the underlying mechanisms remain largely unknown. GROWTH DIFFERENTIATION FACTOR 15 GDF15 is increased in AIRWAY EPITHELIUM of COPD SMOKERS and CS-exposed human AIRWAY EPITHELIAL CELLS but its role in CS-induced AIRWAY EPITHELIAL SENESENCE is unclear. In this study, we first analyzed expression of GDF15 and CELLULAR SENESENCE markers in AIRWAY EPITHELIAL CELLS of current SMOKERS and NONSMOKERS. Second, we determined the role of GDF15 in CS-induced AIRWAY EPITHELIAL SENESENCE by using the CLUSTERED REGULARLY INTERSPACED SHORT PALINDROMIC REPEATS CRISPR associated-9 CAS9 GENOME EDITING approach. Finally, we examined whether EXOGENOUS GDF15 PROTEIN promoted AIRWAY EPITHELIAL SENESENCE through the ACTIVIN RECEPTOR-LIKE KINASE 1 ALK1 SMAD1 PATHWAY GDF15 UP-REGULATION was found in parallel with increased CELLULAR SENESENCE markers P21 P16 and HIGH MOBILITY GROUP box 1 HMGB1 in AIRWAY EPITHELIAL CELLS of current SMOKERS compared with NONSMOKERS. Moreover, CS extract CS-E induced CELLULAR SENESENCE in cultured human AIRWAY EPITHELIAL CELLS, represented by induced SENESENCE-ASSOCIATED B-GALACTOSIDASE activity, INHIBITED CELL PROLIFERATION, increased P21 expression, and increased release of HMGB1 and IL-6. Disruption of GDF15 significantly INHIBITED CS-E-induced AIRWAY EPITHELIAL SENESENCE. Lastly, GDF15 PROTEIN BOUND to the ALK1 RECEPTOR and promoted AIRWAY EPITHELIAL SENESENCE via activation of the SMAD1 PATHWAY. Our findings highlight an important contribution of GDF15 in promoting AIRWAY EPITHELIAL SENESENCE upon CS exposure. SENESENT AIRWAY EPITHELIAL CELLS that CHRONICALLY accumulate in CS-exposed LUNGS could contribute substantially to CHRONIC AIRWAY INFLAMMATION in COPD development and progression.

SMAD1

Normalized: Mothers against decapentaplegic homolog 1

Domains: Biology, Engineering

conf: 0.7665

Mothers against decapentaplegic homolog 1 also known as SMAD family member 1 or SMAD1 is a protein that in humans is encoded by the SMAD1 gene.

Entrez Gene ID	4086
HGNC gene symbol	SMAD1
HGNC ID	6767
OMIM ID	601595
subclass of	Q20747295
Ensembl Gene ID	ENSG00000170365
HomoloGene ID	21196
RefSeq RNA ID	XM_011531964
RefSeq RNA ID	XM_011531964
RefSeq RNA ID	XM_011531964

- entity recognition and disambiguation against Wikidata at document level, for example a PDF with layout positioning and structure-aware annotations,

with previous failed attempts at cardiac resynchronization therapy

Anoop K. Shetty^{1,2*}, Simon G. Duckett^{1,2}, Julian Bostock¹, Eric Rosenthal¹, and C. Aldo Rinaldi^{1,2}

¹Cardiothoracic Department, Guys and St Thomas' Hospital NHS Foundation Trust, London, UK; and ²King's College London, Westminster Bridge Road, London SE17EH, UK
Received 22 September 2010; accepted after revision 17 January 2011; online publish-ahead-of-print 22 February 2011

Alms

Problems with implanting a left ventricular LV lead during cardiac resynchronization therapy CRT procedures are not uncommon and may occur for a variety of reasons including phrenic nerve stimulation PNS and high capture thresholds. We aimed to perform successful CRT in patients with previous LV lead problems using the multiple pacing configurations available with the St Jude Quartet model 1458Q quadripolar LV lead to overcome PNS or high capture thresholds.

Methods and results

Four patients with previous failed attempts at LV lead implantation underwent a further attempt at CRT using a Quartet lead. In all four cases, successful CRT was achieved using a Quartet lead placed in a branch of the coronary sinus. Problems with PNS or high capture thresholds were seen in all four patients but were successfully overcome. Satisfactory lead parameters were seen at implant, pre-discharge, and at short-term follow-up 8.5 ± 5 weeks.

Conclusion

The Quartet lead allows 10 different pacing vectors to be used and may overcome common pacing problems because of the multiple pacing configurations available. Problems with either PNS or unsatisfactory pacing parameters experienced during CRT may be resolved simply by changing the pacing configuration using this quadripolar lead system.

Keywords

CRT • Quadripolar lead • Phrenic nerve stimulation PNS • Failed implant

Introduction

Cardiac resynchronization therapy CRT improves heart failure symptoms and reduces hospitalizations and risk of death in patients with left ventricular LV dysfunction and a broad QRS.^{1–3} Failure to implant an LV lead during attempted CRT occurs in ~5–15% of cases.^{4–6} This may be because of an inability to cannulate the coronary sinus CS ostium, an inability to pass the LV lead into a CS branch, unsatisfactory pacing parameters, or phrenic nerve stimulation PNS. In a study of 197 consecutive patients undergoing CRT, Biffi et al. showed that

clinically relevant PNS occurred in 22% of patients at CRT implant or follow-up and that its occurrence was highest in those patients for whom the LV lead was placed at pacing sites most associated with reverse remodelling. In the aforementioned study, 7% of patients required an LV lead revision or CRT to be turned off and cathodal programmability the capability to program either the proximal or the distal LV lead electrode as cathode was described as a possible solution to the problem of PNS.⁷ Indeed, other studies^{8,9} have shown that the availability of multiple pacing configurations may overcome problems with high pacing capture thresholds and PNS.

* Corresponding author. Tel: +44 20 7188 8376; fax: +44 20 7188 5442. Email: anoop.shetty@kcl.ac.uk
Published on behalf of the European Society of Cardiology. All rights reserved. © The Author 2011. For permissions please email: journals.permissions@oup.com.

- search query disambiguation (the short text mode) - below disambiguation of the search query “concrete pump sensor” in the service test console,

CARDIAC RESYNCHRONIZATION THERAPY




Normalized: Cardiac resynchronization therapy





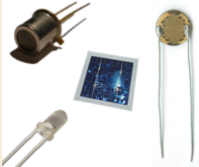

Domains: Surgery

conf: 0.791

An implanted cardiac resynchronization device is a medical device used in cardiac resynchronization therapy CRT. It resynchronizes the contractions of the heart's ventricles by sending tiny electrical impulses to the heart muscle, which can help the heart pump blood throughout the body more efficiently. CRT defibrillators CRT-D also incorporate the additional function of an implantable cardioverter-defibrillator, to quickly terminate an abnormally fast, life-threatening heart rhythm. CRT and CRT-D have become increasingly important therapeutic options for patients with moderate and severe heart failure.

Commons category	Implanted cardiac resynchronization device
JSTOR topic ID	cardiac-resynchronization-therapy

References:   

concrete	<p>Conf: 0.3606</p> <p>Concrete is a composite material composed of fine and coarse aggregate bonded together with a fluid cement (cement paste) that hardens (cures) over time. In the past lime cement binders were often used, such as lime putty, but sometimes with other hydraulic cement, such as a calcium aluminat cement or with Portland cement to form Portland cement concrete (for its visual resemblance to Portland stone). Many other non-cementitious types of concrete exist with other methods of binding aggregate together, including asphalt concrete with a bitumen binder, which is frequently used for road surface, and polymer concrete that use polymers as a binder.</p>		
concrete pump	<p>Conf: 0.9478</p> <p>A concrete pump is a machine used for transferring liquid concrete by pump. There are two types of concrete pumps. The first type of concrete pump is attached to a truck or longer units are on semi-trailers. It is known as a boom concrete pump because it uses a remote-controlled articulating robot arm (called a <i>boom</i>) to place concrete accurately. Boom pumps are used on most of the larger construction projects as they are capable of pumping at very high volumes and because of the labour saving nature of the placing boom. They are a revolutionary alternative to line-concrete pumps.</p>		
pump	<p>Conf: 0.327</p> <p>A pump is a device that moves fluids (liquid or gas), or sometimes slurries, by mechanical action, typically converted from electrical energy into Hydraulic energy. Pumps can be classified into three major groups according to the method they use to move the fluid: <i>direct lift</i>, <i>displacement</i>, and <i>gravity</i> pumps. Pumps operate by some mechanism (typically reciprocating or rotary), and consume energy to perform mechanical work moving the fluid. Pumps operate via many energy sources, including manual operation, electricity, engines, or wind power, and come in many sizes, from microscopic for use in medical applications, to large industrial pumps.</p>		
sensor	<p>Conf: 0.3661</p> <p>In the broadest definition, a sensor is a device, module, machine, or subsystem whose purpose is to detect events or changes in its environment and send the information to other electronics, frequently a computer processor. A sensor is always used with other electronics.</p>		

- weighted term vector disambiguation (a term being a phrase),

Supervised machine learning is used for the disambiguation, based on Random Forest and Gradient Tree Boosting exploiting various features. The main disambiguation techniques include graph distance to measure word and entity relatedness and distributional semantic distance based on word and entity embeddings. Training is realized exploiting Wikipedia, which offers for each language a wealth of usage data about entity mentions in context. Results include in particular Wikidata identifiers and, optionally, statements.

The API uses a full Query DSL with many customization capacities. It offers for instance the possibility to apply filters based on Wikidata properties and values, allowing to create specialised entity identification and extraction (e.g. extract only taxon entities or only medical entities in a document) relying on million entities and statements present in Wikidata.

The tool currently supports 11 languages, English, French, German, Spanish, Italian, Arabic, Japanese, Chinese (Mandarin), Russian, Portuguese and Farsi. For English and French, a Name Entity Recognition based on CRF [grobid-ner](#) is used in combination with the disambiguation. For each recognized entity in one language, it is possible to complement the result with crosslingual information in the other languages. A *nbest* mode is available. Domain information are produced for a large amount of entities in the technical and scientific fields, together with Wikipedia categories and confidence scores.

The tool is developed in Java and has been designed for fast processing (at least for a NERD system, around 1000-2000 tokens per second on a medium-profile linux server single thread or one PDF page of a scientific articles in less than 1 second), with limited memory (at least for a NERD system, here 3GB of RAM as minimum) and to offer relatively close to state-of-the-art accuracy (more to come!). A search query can be disambiguated in 1-10 milliseconds. *entity-fishing* uses the very fast [SMILE ML](#) library for machine learning and a [JNI integration of LMDB](#) as embedded database.

1.4 How to cite

If you want to cite this work, please refer to the present GitHub project, together with the [Software Heritage](<https://www.softwareheritage.org/>) project-level permanent identifier. For example, with BibTeX:

```
@misc{entity-fishing,
  title = {entity-fishing},
  howpublished = {\url{https://github.com/kermitt2/entity-fishing}},
  publisher = {GitHub},
  year = {2016--2022},
  archivePrefix = {swh},
  eprint = {1:dir:cb0ba3379413db12b0018b7c3af8d0d2d864139c}
}
```

1.5 License and contact

entity-fishing is distributed under [Apache 2.0 license](#). The dependencies used in the project are either themselves also distributed under Apache 2.0 license or distributed under a compatible license.

The documentation is distributed under [CC-0](#) license and the annotated data under [CC-BY](#) license.

If you contribute to *entity-fishing*, you agree to share your contribution following these licenses.

Main author and contact: Patrice Lopez (<patrice.lopez@science-miner.com>)

Install, build, run, and monitor

2.1 Install, build, and run

entity-fishing requires JDK 1.8 or higher. It supports Linux-64. Mac OS environments should work fine, but it is *unofficially* supported. Below, we make available the up-to-date and full binary index data for Linux-64 architecture.

Running the service requires at least 3GB of RAM for processing text inputs, but more RAM will be exploited if available for speeding up access to the compiled Wikidata and Wikipedia data (including Wikidata statements associated to entities) and for enabling high rate parallel processing. In case PDF are processed, a minimum of 8GB is required due to additional PDF parsing and structuring requirements. For parallel processing of PDF exploiting multithreading (e.g. 10 parallel threads), 16GB is recommended.

After decompressing all the index data, up to 100 GB of disk space will be used if you wish to use all the supported languages (en, fr, de, it, es, ar, zh, ru, ja, pt, fa) - be sure to have enough free space. For running English language only, you will need around 50 GB. SSD is highly recommended for best performance and experience, in particular with a low amount of available RAM (e.g. RAM < 4GB).

First install GROBID and `grobid-ner`, see the relative instruction of [GROBID](#) and [grobid-ner](#).

You need to install latest current stable version 0.7.1 of GROBID and `grobid-ner`. For GROBID:

Clone GROBID source code from github, latest stable version (currently 0.7.1):

```
$ git clone https://github.com/kermitt2/grobid.git --branch 0.7.1
```

Then build Grobid, in the main directory:

```
$ cd grobid
$ ./gradlew clean install
```

The path to `grobid-home` shall indicated in the file `data/config/mention.yaml` of the *entity-fishing* project, for instance:

```
# path to the GROBID home (for grobid-ner, grobid, etc.)
grobidHome: ../grobid/grobid-home/
```

For grobid-ner now, under grobid/, install grobid-ner:

```
$ git clone https://github.com/kermitt2/grobid-ner.git
```

Then build grobid-ner, in the sub-project directory:

```
$ cd grobid-ner
$ ./gradlew copyModels
$ ./gradlew clean install
```

Install *entity-fishing*:

```
$ git clone https://github.com/kermitt2/entity-fishing.git
```

Then install the compiled indexed data:

1. Download the zipped data files corresponding to your environment. The knowledge-base (Wikidata, `db-kb.zip`) and the English Wikipedia data (`db-en.zip`) must always been installed as minimal set-up. You can then add your languages of choice at the following links. Total is around 29 GB compressed, and around 90 GB uncompressed. The data for this version 0.0.5 correspond to the Wikidata and Wikipedia dumps from Feb., 1st 2022. The Knowledge Base part contains around 96 million entities. In this available KB data file, only the statements for entities having at least one Wikipedia page in one of the 9 supported languages are loaded (it's possible to load all of them by regenerating the KB with a dedicated parameter).

Linux

- <https://science-miner.s3.amazonaws.com/entity-fishing/0.0.5/db-kb.zip> (7.5 GB)
- <https://science-miner.s3.amazonaws.com/entity-fishing/0.0.5/db-en.zip> (6.9 GB)
- <https://science-miner.s3.amazonaws.com/entity-fishing/0.0.5/db-fr.zip> (2.3 GB)
- <https://science-miner.s3.amazonaws.com/entity-fishing/0.0.5/db-de.zip> (2.6 GB)
- <https://science-miner.s3.amazonaws.com/entity-fishing/0.0.5/db-es.zip> (1.8 GB)
- <https://science-miner.s3.amazonaws.com/entity-fishing/0.0.5/db-it.zip> (1.6 GB)
- <https://science-miner.s3.amazonaws.com/entity-fishing/0.0.5/db-ar.zip> (1.3 GB)
- <https://science-miner.s3.amazonaws.com/entity-fishing/0.0.5/db-zh.zip> (1.3 GB)
- <https://science-miner.s3.amazonaws.com/entity-fishing/0.0.5/db-ru.zip> (2.3 GB)
- <https://science-miner.s3.amazonaws.com/entity-fishing/0.0.5/db-ja.zip> (1.8 GB)
- <https://science-miner.s3.amazonaws.com/entity-fishing/0.0.5/db-pt.zip> (1.8 GB)
- <https://science-miner.s3.amazonaws.com/entity-fishing/0.0.5/db-fa.zip> (1.8 GB)

MacOS is not officially supported and should not be used for production. For convenience, we still make available the MacOS data version 0.0.3 corresponding to the Wikidata and Wikipedia dumps from mid-2018. Although outdated and Arabic not available, they are still compatible with the *entity-fishing* version 0.0.4 and 0.0.5 and could be used for test/development. However, we strongly recommend to use the Linux version for any serious works.

MacOS

- <https://science-miner.s3.amazonaws.com/entity-fishing/0.0.3/macos/db-kb.zip> (4.1 GB)
- <https://science-miner.s3.amazonaws.com/entity-fishing/0.0.3/macos/db-en.zip> (5.5 GB)
- <https://science-miner.s3.amazonaws.com/entity-fishing/0.0.3/macos/db-fr.zip> (1.9 GB)
- <https://science-miner.s3.amazonaws.com/entity-fishing/0.0.3/macos/db-de.zip> (2.0 GB)

- <https://science-miner.s3.amazonaws.com/entity-fishing/0.0.3/macos/db-es.zip> (1.5 GB)
- <https://science-miner.s3.amazonaws.com/entity-fishing/0.0.3/macos/db-it.zip> (1.3 GB)

1. Unzip the db archives files under `data/db/`.

This will install several sub-directories, one per language, plus wikidata (db-kb): `data/db/db-XY/`, with XY equal to `fr`, `en`, `it`, `es`, `en`, `ar`, `zh`, `ru`, `ja`, `pt` and `fa`. The full uncompressed data is more than 90 GB.

2. Build the project, under the *entity-fishing* project repository.

```
$ ./gradlew clean build
```

You should be now ready to run the service.

3. Run the service:

```
$ ./gradlew run
```

The test console is available at port : 8090 by opening in your browser: <http://localhost:8090>

The service port, CORS parameters, and logging parameters can be configured in the file `data/config/service.yaml`.

For more information, see the next section on the *entity-fishing* Console.

2.2 Metrics and monitoring

As the server is started, the Dropwizard administrative/service console can be accessed at <http://localhost:8091/> (default hostname and port)

DropWizard metrics are available at <http://localhost:8091/metrics?pretty=true>

Prometheus metrics (e.g. for Graphana monitoring) are available at <http://localhost:8091/metrics/prometheus>

2.3 Creating a new Knowledge Base version

The knowledge base used by *entity-fishing* can be updated with new versions of Wikidata and Wikipedia using the pre-processing from the library GRISP, see <https://github.com/kermitt2/grisp>.

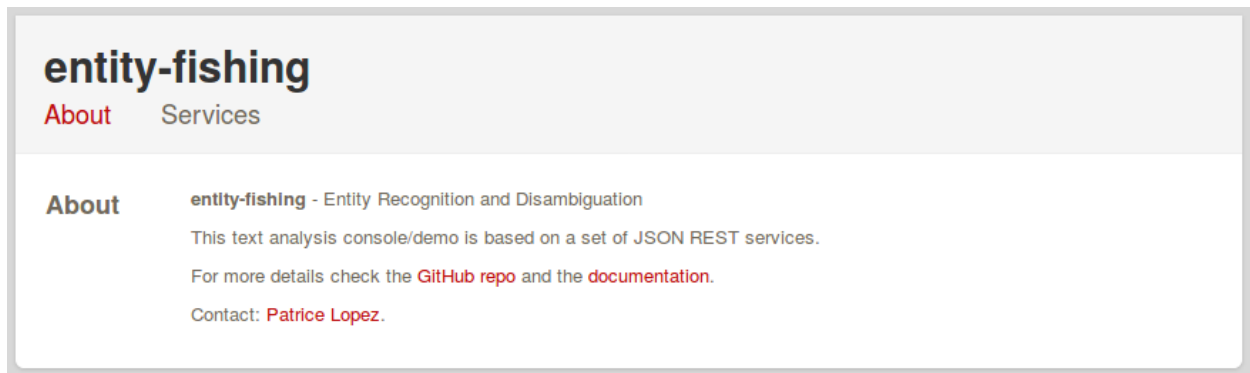
entity-fishing Console

The *entity-fishing* console is a **graphical web interface**, part of the *entity-fishing* project, providing means to discover and test the service. With the console, it is possible to process chunks of text (typically a paragraph), PDF files and to verify which entities are recognised and how they are disambiguated.

The console is also a **reference implementation** in javascript (with JQuery) of a web application using the *entity-fishing* API service. As such, it illustrates how to call the services with mainstream Ajax queries, how to parse JSON results with *vulgus JQuery* and how to dynamically annotate a PDF with PDF.js and a dynamic HTML layer.

The console is available at the root address of the server (e.g. `http://localhost:8090` by default, to be changed in the configuration file `data/config/service.yaml`).

The About page provides licence (Open Source Apache 2 licence for the entire tool including used dependencies) and contact information.



The web page *Services* allows to test the different REST requests.

[About](#)
[Services](#)

Service to call

disambiguate - text

```
{
  "text": "Mexico: Recovery excitement brings Mexican markets to life.\n
Henry Tricks\n
Mexico City\n
Emerging evidence that Mexico's economy was back on the recovery track sent Mexican markets into a buzz of excitement Tuesday, with stocks closing at record highs and interest rates at 19-month lows.\n
Mexico has been trying to stage a recovery since the beginning of this year and it's always been getting ahead of itself in terms of fundamentals," said Matthew Hickman of Lehman Brothers in New York.\n
"
```

Submit

Annotations

Response

MEXICO: Recovery excitement brings MEXICAN markets to life.

Henry Tricks

MEXICO CITY

Emerging evidence that MEXICO's economy was back on the recovery track sent MEXICAN markets into a buzz of excitement TUESDAY, with stocks closing at record highs and interest rates at 19-month lows.

MEXICO has been trying to stage a recovery since the beginning of this year and it's always been getting ahead of itself in terms of fundamentals," said MATTHEW HICKMAN of LEHMAN BROTHERS in NEW YORK.


"Now we're at the point where the fundamentals are with us. The history is now falling out of view."

That history is one etched into the minds of all investors in MEXICO: an economy in crisis since DECEMBER 1994, a free-falling PESO and stubbornly high interest rates.

This week, however, second-quarter GROSS DOMESTIC PRODUCT was reported up 7.2 PERCENT, much

GROSS DOMESTIC PRODUCT

Normalized: **Gross domestic product**
Domains: **Finance**
conf: 0.832



Gross domestic product (GDP) is a monetary **measure** of the market value of all final goods and services produced in a period (quarterly or yearly) of **time**. **Nominal GDP** estimates are commonly used to determine the economic performance of a whole country or region, and to make international comparisons. **Nominal GDP per capita** does not, however, reflect differences in the **cost of living** and the **inflation rates** of

A text form allows the analysis of any queries expressed in the *entity-fishing* query DSL (see next section). On the right side of the input form, samples of text can be found, from scientific articles, news and historical documents in supported various languages.

In the lower part, entities are recognised in the provided text and displayed using different colors, based on the entity type and domain. On the lower right side, an infobox is displaying information provided by the service about the disambiguated Wikidata/Wikipedia entity.

In this example the text box is used to disambiguate a search query:

12

Chapter 3. *entity-fishing* Console

Service to call
disambiguate - text

```







{
  "text": "",
  "shortText": "concrete pump sensor",
  "termVector": [],
  "language": {
    "lang": "en"
  }
}

```

Submit

WW1 Reuters_1
PubMed_1 Reuters_2
PubMed_2 French_1
HAL_1 German_1
Italiano_1 Spanish_1

Entitles
Response

concrete	<p>Conf: 1</p> <p>Concrete is a composite material composed of coarse aggregate bonded together with a fluid cement that hardens over time. Most concretes used are lime-based concretes such as Portland cement concrete or concretes made with other hydraulic cement, such as ciment fondu. However, asphalt concrete, which is frequently used for road surface, is also a type of concrete, where the cement material is bitumen, and polymer concrete are sometimes used where the cementing material is a polymer. When aggregate is mixed together with dry Portland cement and water, the mixture forms a fluid slurry that is easily poured and molded into shape. The cement reacts chemically with the water and other ingredients to form a hard matrix that binds the materials together into a durable stone-like material that has many uses. Often, additives (such as pozzolan or superplasticizer) are included in the mixture to improve the physical properties of the wet mix or the finished material. Most concrete is poured with reinforcing materials (such as rebar) embedded to provide tensile strength, yielding reinforced concrete.</p>		
concrete pump	<p>Conf: 1</p> <p>A concrete pump is a machine used for transferring liquid concrete by pump. There are two types of concrete pumps. The first type of concrete pump is attached to a truck or longer units are on semi-trailers. It is known as a boom concrete pump because it uses a remote-controlled articulating robot arm (called a <i>boom</i>) to place concrete accurately. Boom pumps are used on most of the larger construction projects as they are capable of pumping at very high volumes and because of the labour saving nature of the placing boom. They are a revolutionary alternative to line-concrete pumps.</p>		
pump	<p>Conf: 1</p> <p>A pump is a device that moves fluids (liquid or gas), or sometimes slurries, by mechanical action. Pumps can be classified into three major groups according to the method they use to move the fluid: <i>direct lift</i>, <i>displacement</i>, and <i>gravity</i> pumps.</p>		

The console allows to test all the different services provided by *entity-fishing*, e.g. it's possible to visualise the various sentences identified by the the sentence segmentation service (more details on this specific service in the REST API documentation).

Service to call

disambiguate - text

"wikimedia"
},
"nbest": false,
"sentence": true,
"customisation": "generic"
}

Submit

WW1

Reuters_1

PubMed_1

Reuters_2

PubMed_2

French_1

HAL_1

German_1

Italiano_1

Spanish_1

Annotations
Response

2 But the failed Russian invasion, causing the fresh German troops to move to the east, allowed the tactical Allied victory at the First Battle of the Marne.
3 Unfortunately for the Allies, the pro-German King Constantine I dismissed the pro-Allied government of E. Venizelos before the Allied expeditionary force could arrive.
4 Beginning in 1915, the Italians under Cadorna mounted eleven offensives on the Isonzo front along the Isonzo River, northeast of Trieste.
5 At the Siege of Maubeuge about 40000 French soldiers surrendered, at the battle of Galicia Russians took about 100-120000 Austrian captives, at the Brusilov Offensive about 325 000 to 417 000 Germans and Austrians surrendered to Russians, at the Battle of Tannenberg 92,000 Russians surrendered.
6 After marching through Belgium, Luxembourg and the Ardennes, the German Army advanced, in the latter half of August, into northern France where they met both the French army, under Joseph Joffre, and the British Expeditionary Force, under Sir John French.

At the SIEGE OF MAUBEUGE about 40000 FRENCH soldiers surrendered, at the BATTLE OF GALICIA RUSSIANS took about 100 120000 AUSTRIAN captives, at the BRUSILOV OFFENSIVE about 325 000 to 417 000 GERMANS and AUSTRIANS surrendered to RUSSIANS, at the BATTLE OF TANNENBERG 92,000 RUSSIANS surrendered.

BATTLE OF TANNENBERG

Type: EVENT
Domains: Administration, Military
conf: 0.8996

References: W

In addition, it is possible to view the service raw response (in JSON format) for helping the integration phase:

Mexico: Recovery excitement brings Mexican markets to life.

Submit

Cendari

Reuters_1

PubMed_1

Reuters_2

PubMed_2

French_1

HAL_1

German_1

Annotations
Response

```

{
  "runtime": 38,
  "onlyNER": false,
  "nbest": false,
  "text": "Mexico: Recovery excitement brings Mexican markets to life.",
  "language": {
    "lang": "en",
    "conf": 0.9999977565831784
  },
  "entities": [
    {
      "rawName": "Mexico",
      "type": "LOCATION",
      "offsetStart": 0,
      "offsetEnd": 6,
      "nerd_score": "0.47886873578089756",
      "nerd_selection_score": "0.8665133879482593",
      "sense": {
        "fineSense": "country/N1"
      }
    }
  ],
  "wikipediaExternalRef": "3966054".

```

More details about the response in the next section.

entity-fishing REST API

As RESTful web services, *entity-fishing* is defined by a certain number of stateless transformations of data made available to “consumers” via its interface.

All these RESTful services are available through Cross-origin resource sharing (CORS), allowing clients, such as web browser and server to interact in a flexible manner with cross-origin request.

4.1 *entity-fishing* query processing

The *entity-fishing* query processing service takes as input a JSON structured query and returns the JSON query enriched with a list of identified and, when possible, disambiguated entities.

The *entity-fishing* service can be applied on 4 types of input content:

- **text**, provided as JSON string value, for example one or several paragraphs of natural language,
- **search query**, provided as JSON string value, corresponding to several search terms used together and which can possibly be disambiguated when associated,
- **weighted vector of terms**, provided by a structured JSON array, where each term will be disambiguated, when possible, in the context of the complete vector - weighted vector of term is a very common structure used in information retrieval, clustering and classification.
- **PDF document**, provided as multipart data with the JSON query string.

One and only one input type is mandatory in a query, otherwise an HTTP error 400 is returned (see response status codes below). Combining multiple inputs in a single request is currently not supported.

4.1.1 Supported languages

In the current version 11 languages are supported: English, French, German, Spanish, Italian, Arabic, Japanese, Chinese (Mandarin), Russian, Portuguese and Farsi are supported. We plan to extend the support in future releases, as long the volume of the Wikipedia corpus for a new language is sufficient.

The service returns an HTTP error 406 if the language of the text to be processed is not supported, see below.

4.1.2 Response status codes

In the following table are listed the status codes returned by this entry point.

HTTP Status code	Reason
200	Successful operation.
400	Wrong request, missing parameters, missing header
404	Indicates property was not found
406	The language is not supported
500	Indicate an internal service error

4.1.3 REST query

POST /disambiguate

(1) Parameters

required	name	content-type value	description
required	query	multipart/form-data	Query to be processed in JSON UTF-8
optional	file	multipart/form-data	PDF file (as multipart)

NOTE: To process the text query only (no PDF), is also possible to send it as normal *application/json* raw data.

(2) Request header

required	name	value	description
optional	Accept	application/json	Set the response type of the output

4.1.4 Query format description

The *entity-fishing* query processing service always consumes a parameter which is a JSON string representing a query, and optionally a PDF file. The service thus follows a Query DSL approach (like, for instance, ElasticSearch) to express queries instead of multiples HTTP parameters. This approach allows queries which are much richer, flexible and simple to express, but also interactive scenarios (where output of the services can be used easily as input after some changes from the user, as for instance in an interactive text editing task).

The JSON query indicates what is the textual content to process, the various (optional) parameters to consider when processing it, optionally some already existing disambiguated entities (already disambiguated by a user or via a particular workflow), and an optional customisation to provide more context to the disambiguation process.

The JSON query is similar to the response of the *entity-fishing* service, so that a *entity-fishing* service response can be sent as query after light modifications in an interactive usage scenario, or to be able to process easily already partially annotated text.

When annotations are present in the query, the *entity-fishing* system will consider them certain and:

- ensure that the user annotations will be present in the output response without inconsistencies with other annotations,
- exploit the user annotations to improve the context for identifying and disambiguating the other possible entities.

Similarly,

- if no language is indicated (usual scenario), the *entity-fishing* service will use a language identifier to detect the correct language and the language resources to use. However, the query can also optionally specify a language for the text to be processed. This will force the service to process the text with the corresponding particular language resources.
- it is possible also to pass an existing sentence segmentation to the *entity-fishing* service via the JSON query, in order that the service provides back identified entities following the given sentence segmentation.

The client must respect the JSON format of the *entity-fishing* response as new query, as described below:

Generic format

The JSON format for the query parameter to be sent to the service is identical to a response of the service:

```
{
  "text": "The text to be processed.",
  "shortText": "term1 term2 ...",
  "termVector": [
    {
      "term": "term1",
      "score": 0.3
    },
    {
      "term": "term2",
      "score": 0.1
    }
  ],
  "language": {
    "lang": "en"
  },
  "entities": [],
  "mentions": ["ner", "wikipedia"],
  "nbest": 0,
  "sentence": false,
  "customisation": "generic",
  "processSentence": [],
  "structure": "grobid"
}
```

One and only one of the 4 possible input type - JSON field text, shortText, termVector or a PDF file - must be provided in a query to be valid. Using multiple input type in the same query is not supported in the version of the API described here.

(1) text

Provides a text to be processed (e.g. one or several paragraphs). The text have be greater than 5 character or 406 is returned. The expected amount of text to disambiguate for the different models is a paragraph (100-150 words). If the amount of text is larger, the text will be automatically segmented into balanced segments of maximum 1000 characters (this default size can be changed), using end-of-line and then sentence boundaries. A sliding context will be managed to pass the previous accumulated context (best entities, identified acronyms, ...) to the following segments.

(2) shortText

Provides a search query to be processed.

(3) termVector

Provides a list of terms, each term being associated to a weight indicating the importance of the term as compared to the other terms.

(4) language

If this field is empty, a language identifier is used. When the source language (parameters language) is pre-set the language is considered certain, and a language identifier is not used.

(5) mentions

Provides the methods to be used to identify mentions to be disambiguated. By default, mentions are identified with an NER (the mentions are all Named Entity found in the input text to be processed), noted `ner` and with all the labels of Wikipedia for the appropriate language (all the anchors and titles used to refer to a Wikipedia page), noted `wikipedia`. The order of the mention identification methods matters.

If the mentions field is an empty array (`"mentions": []`), only the mentions present in the field `entities` will be disambiguated. This case allows to target the disambiguation only to one or a few mentions in a sentence or a text.

(6) entities

In the input example above, the list `entities` can be used to provide predefined entities or mentions (typically pre-annotated by a user). Having an already annotated entity helps the disambiguation service to resolve entity mentions by offering an important contribution to the global context. When the entities attribute is not present or empty there are simply no predefined annotations.

For example having a text with the mention “Washington” and manually providing its referring entity (e.g. the city Washington DC) is an important advantage for a correct disambiguation of the other entity mentions in the text.

Below an example of how the pre-annotated entity can be provided. The algorithm would naturally disambiguate *German Army* with *German Army (Wehrmacht)* (wikipediaId: 12354993) because the text is contextualised on the First World War. The users can alter this result, by forcing the term to be the *German Army* of the Second World War (wikipediaId: 11702744). In the response the entity should be returned with confidence 1.0 (as it has been manually provided).

In order to get the wikipedia information for a term, check the [term lookup documentation](#).

NOTE: At the moment the entity is taken in account only when the `wikipediaExternalRef` is provided:

```
{
  "text": "Austria invaded and fought the Serbian army at the Battle of Cer and_
↪Battle of Kolubara beginning on 12 August.",
  "language": {
    "lang": "en"
  },
  "entities": [
    {
      "rawName": "German Army",
      "offsetStart": 1107,
      "offsetEnd": 1118,
      "wikipediaExternalRef": 11702744,
      "wikidataId": "Q701923"
```

(continues on next page)

(continued from previous page)

```

    }
  ]
}

```

In a typical interactive scenario, an application client first sends a text to be processed via the `/disambiguate` service, and receives a JSON response with some entities. The annotated text is displayed to a user which might correct some invalid annotations. The client updates the modified annotations in the first JSON response and can send it back to the service now as new query via the `/disambiguate`. The corrected annotations will then be exploited by the *entity-fishing* system to possibly improve the other annotations and disambiguations.

The `entities` field can also contains only mentions defined by their offsets in the text, without wikidata/wikipedia information. The mention will then be considered as a forced target mention to be disambiguated. In case the above `mentions` field (5) is an empty array (i.e. no method to detect mention), these mentions defined in `entities` will still be considered and disambiguated. This a way to limit the disambiguation to one or few mentions in a text, with significant runtime gain.

(7) processSentence

The `processSentence` parameter is introduced to support interactive text editing scenarios. For instance, a user starts writing a text and wants to use the *entity-fishing* service to annotate dynamically the text with entities as it is typed.

To avoid having the server reprocessing several time the same chunk of text and slowing down a processing time which has to be almost real time, the client can simply indicate a sentence - the one that has just been changed - to be processed.

The goal is to be able to process around two requests per second, even if the typed text is very long, so that the annotations can be locally refreshed smoothly, even considering the fastest keystroke rates that a human can realize.

The `processSentence` parameter is followed by a list of notations (only numbers in integer, e.g. `[1, 7]` - note that the index starts from 0) corresponding to the sentence index will limit the disambiguation to the selected sentences, while considering the entire text and the previous annotations.

In this example only the second sentence will be processed by *entity-fishing*:

```

{
  "text": "The army, led by general Paul von Hindenburg defeated Russia in a series_
↳of battles collectively known as the First Battle of Tannenberg. But the failed_
↳Russian invasion, causing the fresh German troops to move to the east, allowed the_
↳tactical Allied victory at the First Battle of the Marne.",
  "processSentence": [
    1
  ]
}

```

When `processSentence` is set, the sentence segmentation is triggered anyway and the value of the attribute `sentence` is ignored:

```

{
  "text": "The army, led by general Paul von Hindenburg defeated Russia in a series_
↳of battles collectively known as the First Battle of Tannenberg. But the failed_
↳Russian invasion, causing the fresh German troops to move to the east, allowed the_
↳tactical Allied victory at the First Battle of the Marne.",
  "processSentence": [
    1
  ],
  "sentences": [

```

(continues on next page)

(continued from previous page)

```

    {
      "offsetStart": 0,
      "offsetEnd": 138
    },
    {
      "offsetStart": 138,
      "offsetEnd": 293
    }
  ],
  "entities": [
    {
      "rawName": "Russian",
      "type": "NATIONAL",
      "offsetStart": 153,
      "offsetEnd": 160
    }
  ]
}

```

Example using CURL (using the query above):

```

curl 'http://cloud.science-miner.com/nerd/service/disambiguate' -X POST -F "query={
  ↳'text': 'The army, led by general Paul von Hindenburg defeated Russia in a series_
  ↳of battles collectively known as the First Battle of Tannenberg. But the failed_
  ↳Russian invasion, causing the fresh German troops to move to the east, allowed the_
  ↳tactical Allied victory at the First Battle of the Marne.', 'processSentence': [ 1_
  ↳], 'sentences': [ { 'offsetStart': 0, 'offsetEnd': 138 }, { 'offsetStart': 138,
  ↳'offsetEnd': 293 } ], 'entities': [ { 'rawName': 'Russian', 'type': 'NATIONAL',
  ↳'offsetStart': 153, 'offsetEnd': 160 } ] }"

```

(8) structure

The **structure** parameter is only considered when the input is a PDF. For processing scientific and technical documents, in particular scholar papers, the value should be **grobid** which is a state of the art tool for structure the body of a scientific paper - it will avoid labelling bibliographical callout (like *Romary and al.*), running foot and head notes, figure content, it will identify the useful areas (header, paragraphs, captions, etc.), handling multiple columns, hyphen, etc. It will apply custom processing based on the nature of the identified structure. This enables “structure-aware” annotations. If no **structure** value is provided, the value **grobid** will be used.

If you wish to process the whole document without specific structure analysis - this is advised for non-scientific papers -, use the value **full**.

Example using CURL for processing the full content of a PDF, *without* preliminar structure recognition:

```

curl 'http://cloud.science-miner.com/nerd/service/disambiguate' -X POST -F "query={
  ↳'language': { 'lang': 'en' } }, 'entities': [], 'nbest': false, 'sentence': false,
  ↳'structure': 'full' }" -F "file=@PATH_FILENAME.pdf"

```

Additional optional parameters

In addition to the different parameters described previously, it is also possible to set *per query* three additional parameters:

- `ngramLength`: the maximum length of a term to be considered as mention, default is 6 (i.e. complex terms will be considered up to 6 words)
- `targetSegmentSize`: the maximum length of a segment to be considered when processing long texts in number of characters, default is 1000 (i.e. a text of 10,000 characters will be segmented in approximatively ten balanced segments of a maximum 1000 characters)
- `minSelectorScore`: this overrides the `minSelectorScore` indicated in the language-specific configuration files. It indicates the minimum score produced by the selector model under which the entities will be pruned. This parameter can be used to modify the balance between precision and recall of the entity recognition.
- `maxTermFrequency`: this overrides the `maxTermFrequency` indicated in the language-specific configuration files. This parameter indicates the maximum term frequency above which the terms will be skipped and not used in the disambiguation. The frequency is expressed as Zipf, i.e. a number typically between 0 and 8. Decreasing the value of this parameter can be used for faster processing runtime of the query, but some entities might be overlooked.

It is advised **not to modify these parameters** in a normal usage of the service, because the different models have been trained with the default parameter values. Modifying these parameters might decrease the accuracy of the service.

The following third additional parameter is currently only used for text queries and relevant to long text:

- `documentLevelPropagation`: if `true`, the entities disambiguated for certain mentions are propagated to other same mentions in the document not labeled with an entity. This allows to maintain a document level consistency where some mentions, due to poorer context, are not disambiguated, while other mentions in richer contexts are disambiguated. To be propagated, the mention **tf-idf** must be higher than a certain threshold in order to propagate only non trivial, minimally discriminant terms. Default is `true`.

PDF input

This service is processing a PDF provided as input after extracting and structuring its raw content. Structuration is currently specialized to scientific and technical articles. Processing a PDF not corresponding to scientific articles is currently not recommended.

In addition to the query, it accepts a PDF file via ``multi-part/form-data``.

The JSON format for the query parameter to be sent to the service is identical to a response of the service:

```
{
  "language": {
    "lang": "en"
  },
  "entities": [],
  "nbest": 0,
  "sentence": false,
  "structure": "grobid"
}
```

An additional parameter related to the processing of the structure of the PDF is available, called *structure*. For processing scientific and technical documents, in particular scholar papers, the value should be *grobid* which is a state of the art tool for structure the body of a scientific paper - it will avoid labelling bibliographical information, foot and head notes, figure content, will identify the useful areas (header, paragraphs, captions, etc.) handling multiple columns, hyphen, etc. and it will apply custom processng based on the identified structure.

If you wish to process the whole document without specific structure analysis (this is advised for non-scientific documents), use the value **full** for the parameter **structure**.

Example using CURL (using the query above):

```
curl 'http://cloud.science-miner.com/nerd/service/disambiguate' -X POST -F "query={
  ↪ 'language': {'lang': 'en'}}, 'entities': [], 'nbest': false, 'sentence': false,
  ↪ 'structure': 'grobid'}" -F "file=@PATH_FILENAME.pdf"
```

Weighted term disambiguation

Process a weighted vector of terms. Each term will be disambiguated - when possible - in the context of the complete vector.

Example request

```
{
  "termVector":
  [
    {
      "term" : "computer science",
      "score" : 0.3
    },
    {
      "term" : "engine",
      "score" : 0.1
    }
  ],
  "language": {
    "lang": "en"
  },
  "nbest": 0
}
```

The termVector field is required for having a well-formed query.

Example using CURL (using the query above):

```
curl 'http://cloud.science-miner.com/nerd/service/disambiguate' -X POST -F "query={
  ↪ 'termVector': [ { 'term' : 'computer science', 'score' : 0.3 }, { 'term' : 'engine',
  ↪ 'score' : 0.1 } ], 'language': { 'lang': 'en' }, 'resultLanguages': ['de'], 'nbest
  ↪ ': 0}"
```

Search query disambiguation

This functionality provides disambiguation for a search query expressed as a “short text”.

The input is the list of terms that are typically provided in the search bar of a search engine, and response time are optimized to remain very low (1-10ms).

For example, let’s consider the search query: “concrete pump sensor”. From this association of search terms, it is clear that the sense corresponding to *concrete* is the material, the entity is the device called *concrete pump*, and it has nothing to do with *concrete* as the antonym of *abstract*.

Processing this kind of input permits to implement semantic search (search based on concept matching) and semantic-based ranking (ranking of documents based on semantic proximity with a query, for instance exploiting classifications, domain information, etc.) in a search engine.

Search query disambiguation uses a special model optimized for a small number of non-strictly ordered terms and trained with search queries.

The difference between standard *text* and *short text* is similar to the one of the [ERD 2014 challenge](#).

It is advised to specify the language of the query terms with the request, because the automatic language detection from short string is more challenging and errors can be relatively frequent.

Example request:

```
{
  "shortText": "concrete pump sensor",
  "language": {
    "lang": "en"
  },
  "nbest": 0
}
```

Example using CURL (using the query above):

```
curl 'http://cloud.science-miner.com/nerd/service/disambiguate' -X POST -F "query={
  → 'shortText': 'concrete pump sensor', 'language': { 'lang': 'en' }, 'nbest': 0}"
```

4.1.5 Response

The response returned by the *entity-fishing* query processing service is basically the same JSON as the JSON query, enriched by the list of identified and, when possible, disambiguated entities, together with a server runtime information.

If the textual content to be processed is provided in the query as a string, the identified entities will be associated to offset positions in the input string, so that the client can associate precisely the textual mention and the entity “annotation”.

If the textual content to be processed is provided as a PDF document, the identified entities will be associated to coordinates positions in the input PDF, so that the client can associate precisely the textual mention in the PDF via a bounding box and makes possible dynamic PDF annotations.

Response when processing a text

```
{
  "software": "entity-fishing",
  "version": "0.0.5",
  "runtime": 34,
  "nbest": false,
  "text": "Austria was attaching Serbia.",
  "language": {
    "lang": "en",
    "conf": 0.9999948456042864
  },
  "entities":
  [
    {
      "rawName": "Austria",
      "type": "LOCATION",
      "offsetStart": 0,
      "offsetEnd": 7,
      "confidence_score": "0.8667510394325003",
      "wikipediaExternalRef": "26964606",
      "wikidataId": "Q40",
      "domains": [
        "Atomic_Physic",
        "Engineering",
        "Administration",

```

(continues on next page)

(continued from previous page)

```

        "Geology",
        "Oceanography",
        "Earth"
    ]
},
[... ] }
```

In the example above, the root layer of JSON values correspond to:

- **runtime**: the amount of time in milliseconds to process the request on server side,
- **nbest**: as provided in the query - when false or 0 returns only the best disambiguated result, otherwise indicates to return up to the specified number of concurrent entities for each disambiguated mention,
- **text**: input text as provided in the query, all the offset position information are based on the text in this field,
- **language**: language detected in the text and his confidence score, if the language is provided in the query then conf is equal to 1.0,
- **entities**: list of entities recognised in the text (with possibly entities provided in the query, considered then as certain),
- **global_categories**: provides a weighted list of Wikipedia categories, in order of relevance that are representing the context of the whole text in input.

For each entity the following information are provided:

- **rawName**: string realizing the entity as it appears in the text
- **offsetStart, offsetEnd**: the position offset of where the entity starts and ends in the text element in characters (JSON UTF-8 characters)
- **confidence_score**: disambiguation and selection confidence score, indicates how certain the disambiguated entity is actually valid for the text mention (this depends a lot on the amount of contextual text where this entity is predicted, the more the better),
- **wikipediaExternalRef**: id of the wikipedia page. This id can be used to retrieve the original page from wikipedia3 or to retrieve all the information associated to the concept in the knowledge base (definition, synonyms, categories, etc. - see the section “Knowledge base concept retrieval”),
- **wikidataId**: the Wikidata QID of the predicted entity. This ID can be used to retrieve the complete Wikidata entry in the knowledge base (the section “Knowledge base concept retrieval”).
- **type**: NER class of the entity (see table of the 27 NER classes below under “2. Named entity types”),

The type of recognised entities are restricted to a set of 27 classes of named entities (see [GROBID NER documentation](#)). Entities not covered by the knowledge bases (the identified entities unknown by Wikipedia) will be characterized only by an entity class and a confidence score, without any reference to a Wikipedia article or domain information.

Response when processing a search query

```

{
  "software": "entity-fishing",
  "version": "0.0.5",
  "runtime": 4,
  "nbest": false,
  "shortText": "concrete pump sensor",
  "language": {
    "lang": "en",
    "conf": 1.0
  },
}
```

(continues on next page)

(continued from previous page)

```

"global_categories":
[
  {
    "weight": 0.08448995135780164,
    "source": "wikipedia-en",
    "category": "Construction equipment",
    "page_id": 24719865
  },
  [...]
],
"entities":
[
  {
    "rawName": "concrete pump",
    "offsetStart": 0,
    "offsetEnd": 13,
    "confidence_score": 0.9501,
    "wikipediaExternalRef": 7088907,
    "wikidataId": "Q786115",
    "domains": [
      "Mechanics",
      "Engineering"
    ]
  },
  {
    "rawName": "sensor",
    "offsetStart": 14,
    "offsetEnd": 20,
    "confidence_score": 0.3661,
    "wikipediaExternalRef": 235757,
    "wikidataId": "Q167676",
    "domains": [
      "Electricity",
      "Electronics",
      "Mechanics"
    ]
  }
]
[...]
```

Response when processing a weighted vector of terms

```

{
  "software": "entity-fishing",
  "version": "0.0.5",
  "date": "2022-06-22T13:21:43.245Z",
  "runtime": 870,
  "nbest": false,
  "termVector": [
    {
      "term": "computer science",
      "score": 0.3,
      "entities": [
        {
          "rawName": "computer science",
          "preferredTerm": "Computer science",
          "confidence_score": 0,
          "wikipediaExternalRef": 5323,
```

(continues on next page)

(continued from previous page)

```

        "wikidataId": "Q21198",
        "definitions": [{
            "definition": "'Computer science' blablabla.",
            "source": "wikipedia-en",
            "lang": "en"
        }]
        "categories": [
            {
                "source": "wikipedia-en",
                "category": "Computer science",
                "page_id": 691117
            },
            [...]
        ],
        "multilingual": [
            {
                "lang": "de",
                "term": "Informatik",
                "page_id": 2335
            }
        ]
    } ]
}
[...]
```

Response description when processing PDF

```

{
  "software": "entity-fishing",
  "version": "0.0.5",
  "date": "2022-06-22T13:29:21.014Z",
  "runtime": 32509,
  "nbest": false,
  "language": {
    "lang": "en",
    "conf": 0.9999987835857094
  },
  "pages": [
    {
      "page_height": 792.0,
      "page_width": 612.0
    },
    {
      "page_height": 792.0,
      "page_width": 612.0
    },
    {
      "page_height": 792.0,
      "page_width": 612.0
    },
    {
      "page_height": 792.0,
      "page_width": 612.0
    }
  ],
  "entities": [
```

(continues on next page)

(continued from previous page)

```

{
  "rawName": "Austria",
  "type": "LOCATION",
  "confidence_score": "0.8667510394325003",
  "pos": [
    { "p": 1, "x": 20, "y": 20, "h": 10, "w": 30 },
    { "p": 1, "x": 30, "y": 20, "h": 10, "w": 30 } ]
  "wikipediaExternalRef": "26964606",
  "wikidataId": "Q40",
  "domains": [
    "Atomic_Physic", "Engineering", "Administration", "Geology", "Oceanography
↪", "Earth"
  ] },
  [...] }

```

As apparent in the above example, for PDF the offset position of the entities are replaced by coordinates information introduced by the JSON attribute `pos`. These coordinates refer to the PDF that has been processed and permit to identify the chunk of annotated text by the way of a list of bounding boxes.

In addition, an attribute `pages` is used to indicate the size of each page of the PDF document which is a necessary information to position correctly annotations.

The next section further specifies the coordinates information provided by the service (see [GROBID](#)).

PDF Coordinates

The PDF coordinates system has three main characteristics:

- contrary to usage, the origin of a document is at the upper left corner. The x-axis extends to the right and the y-axis extends downward,
- all locations and sizes are stored in an abstract value called a PDF unit,
- PDF documents do not have a resolution: to convert a PDF unit to a physical value such as pixels, an external value must be provided for the resolution.

In addition, contrary to usage in computer science, the index associated to the first page is 1 (not 0).

The response of the processing of a PDF document by the *entity-fishing* service contains two specific structures for positioning entity annotations in the PDF:

- the list of page size, introduced by the JSON attribute `pages`. The dimension of each page is given successively by two attributes `page_height` and `page_width`.
- for each entity, a json attribute `pos` introduces a list of bounding boxes to identify the area of the annotation corresponding to the entity. Several bounding boxes might be necessary because a textual mention does not need to be a rectangle, but the union of rectangles (a union of bounding boxes), for instance when a mention to be annotated is on several lines.

A bounding box is defined by the following attributes:

- `p`: the number of the page (beware, in the PDF world the first page has index 1!),
- `x`: the x-axis coordinate of the upper-left point of the bounding box,
- `y`: the y-axis coordinate of the upper-left point of the bounding box (beware, in the PDF world the y-axis extends downward!),
- `h`: the height of the bounding box,
- `w`: the width of the bounding box.

As a PDF document expresses value in abstract PDF unit and do not have resolution, the coordinates have to be converted into the scale of the PDF layout used by the client (usually in pixels). This is why the dimension of the pages are necessary for the correct scaling, taking into account that, in a PDF document, pages can be of different size.

The *entity-fishing* console offers a reference implementation with PDF.js for dynamically positioning entity annotations on a processed PDF.

4.2 Knowledge base concept retrieval

This service returns the knowledge base concept information. In our case case, language-independent information from Wikidata will be provided (Wikidata identifier, statements), together with language-dependent information (all the Wikipedia information: Wikipedia categories, definitions, translangual information, etc.). This service is typically used in pair with the main *entity-fishing* query processing service in order to retrieve a full description of an identified entity.

The service supports the following identifiers:

- wikidata identifier (starting with *Q*, e.g. *Q61*)
- wikipedia identifier

The *entity-fishing* content processing service returns the identifiers of the resulting entities with some position offset information. Then, if the client wants, for instance, to display an infobox for this entity, it will send a second call to this service and retrieve the full information for this particular entity. Adding all the associated information for each entity in the response of the *entity-fishing* query processing service would result in a very large response which would slow a lot the client, such as a web browser for instance. Using such separate queries allows efficient asynchronous calls which will never block a browser and permits to make only one call per entity, even if the same entity has been found in several places in the same text.

The *entity-fishing* console offers an efficient reference implementation with Javascript and Ajax queries through the combination of the main *entity-fishing* query processing service and the Knowledge base concept retrieval.

4.2.1 Response status codes

In the following table are listed the status codes returned by this entry point.

HTTP Status code	Reason
200	Successful operation.
400	Wrong request, missing parameters, missing header
404	Indicates property was not found
500	Indicate an internal service error

GET /kb/concept/{id}

(1) Parameters

re-quired	name	content-type value	description
re-quired	id	String	ID of the concept to be retrieved (wikipedia, wikidata id (starting with <i>Q</i>) or property (starting with <i>P</i>).
op-tional	lang	String	(valid only for wikipedia IDs) The language knowledge base where to fetch the concept from. Default: <i>en</i> .

(2) Request header

required	name	value	description
optional	Accept	application/json	Set the response type of the output

(3) Example response

```
{
  "rawName": "Austria",
  "preferredTerm": "Austria",
  "confidence_score": "0.0",
  "wikipediaExternalRef": "26964606",
  "wikidataId": "Q1234"
  "definitions": [
    {
      "definition": "'Austria', officially the 'Republic of Austria'",
      "source": "wikipedia-en",
      "lang": "en"
    }
  ],
  "categories": [
    {
      "source": "wikipedia-en",
      "category": "Austria",
      "page_id": 707451
    },
    {
      "lang": "de",
      "source": "wikipedia-en",
      "category": "Erasmus Prize winners",
      "page_id": 1665997
    }
  ],
  "multilingual": [
    {
      "lang": "de",
      "term": "Österreich",
      "page_id": 1188788
    },
    {
      "lang": "fr",
      "term": "Autriche",
      "page_id": 15
    }
  ]
}
```

The elements present in this response are:

- **rawName**: The term name
- **preferredTerm**: The normalised term name
- **confidence_score**: always 0.0 because no disambiguation took place in a KB access
- **wikipediaExternalRef**: unique identifier of the concept in wikipedia
- **wikidataId**: unique identifier of the concept in wikidata

- **definitions:** list of wikipedia definitions (usually in wikipedia a concept contains one and only one definition). Each definition is characterized by three properties:
- **definition:** The text of the definition
- **source:** The knowledge base from which the definition comes from (in this case can be wikipedia-en, wikipedia-de and wikipedia-fr)
- **lang:** the language of the definition
- **categories:** This provides a list of Wikipedia categories⁷ directly coming from the wikipedia page of the disambiguated entity. Each category is characterised by the following properties:
- **category:** The category name
- **source:** The knowledge base from which the definition comes from.
- **pageId:** the Id of the page describing the category
- **domains:** For each entry, Wikipedia provides a huge set of categories, that are not always well curated (1 million categories in the whole wikipedia). Domains are generic classification of concepts, they are mapped from the wikipedia categories.
- **multilingual:** provides references to multi-languages resources referring to the same entity. E.g. the entity country called Austria is Österreich in German wikipedia and Autriche in French wikipedia. The page_id provided here relates to the language-specific Wikipedia (e.g. in the above example the page_id for the country Autriche in the French Wikipedia is 15).

4.3 Term Lookup

This service is used to search terms in the knowledge base. This service is useful to verify how many ambiguity a certain term can generate.

4.3.1 Response status codes

In the following table are listed the status codes returned by this entry point.

GET /kb/term/{term}

(1) Parameters

re-quired	name	content-type value	description
required	term	String	The term to be retrieved
optional	lang	String	The language knowledge base where to fetch the term from. Default: <i>en</i> .

(2) Request header

required	name	value	description
optional	Accept	application/json	Set the response type of the output

4.4 Language identification

Identify the language of a provided text, associated to a confidence score.

4.4.1 Response status codes

In the following table are listed the status codes returned by this entry point.

HTTP Status code	Reason
200	Successful operation.
400	Wrong request, missing parameters, missing header
404	Indicates property was not found
500	Indicate an internal service error

POST /language

(1) Parameters

required	name	content-type value	description
required	text	String	The text whose language needs to be identified

(2) Request header

required	name	value	description
optional	Accept	application/json	Set the response type of the output
optional	Content-Type	multipart/form-data	Define the format of the posted property

(3) Example response (ISO 639-1)

Here a sample of the response

```
{
  "lang": "en",
  "conf": 0.9
}
```

GET /language?text={text}

(1) Parameters

required	name	content-type value	description
required	text	String	The text whose language needs to be identified

(2) Request header

required	name	value	description
optional	Accept	application/json	Set the response type of the output

(3) Example response (ISO 639-1)

Here a sample of the response

```
{
  "lang": "en",
  "conf": 0.9
}
```

4.5 Sentence segmentation

This service segments a text into sentences. It is useful in particular for the interactive mode for indicating that only certain sentences need to be processed for a given query.

Beginning and end of each sentence are indicated with offset positions with respect to the input text.

4.5.1 Response status codes

In the following table are listed the status codes returned by this entry point.

POST /segmentation

(1) Parameters

required	name	content-type value	description
required	text	String	The text to be segmented into sentences

(2) Request header

required	name	value	description
optional	Accept	application/json	Set the response type of the output
optional	Content-Type	multipart/form-data	Define the format of the posted property

(3) Example response

Here a sample of the response

```
{
  "sentences": [
    {
      "offsetStart": 0,
      "offsetEnd": 7
    },
    {
      "offsetStart": 6,
      "offsetEnd": 21
    }
  ]
}
```

GET /segmentation?text={text}

(1) Parameters

required	name	content-type value	description
required	text	String	The text whose language needs to be identified

(2) Request header

required	name	value	description
optional	Accept	application/json	Set the response type of the output

(3) Example response

Here a sample of the response:

```
{
  "sentences": [
    {
      "offsetStart": 0,
      "offsetEnd": 7
    },
    {
      "offsetStart": 6,
      "offsetEnd": 21
    }
  ]
}
```

4.6 Customisation API

The customisation is a way to specialize the entity recognition, disambiguation and resolution for a particular domain. This API allows to manage customisations for the *entity-fishing* instance which can then be used as a parameter by the *entity-fishing* services.

Customisation are identified by their name (or, also called profile in the API).

4.6.1 Customisation body

The JSON profile of a customisation to be sent to the server for creation and extension has the following structure:

```
{
  "wikipedia": [
    4764461,
    51499,
    1014346
  ],
  "language": { "lang": "en" },
  "texts": [
    "World War I (WWI or WW1 or World War One), also known as Germany and Austria-↪Hungary."
  ],
  "description": "Customisation for World War 1 domain"
}
```

The context will be build based on Wikipedia articles and raw texts, which are all optional. Wikipedia articles are expressed as an array of Wikipedia page IDs.

Texts are represented as an array of raw text segments.

4.6.2 Response status codes

In the following table are listed the status codes returned by this entry point.

HTTP Status code	Reason
200	Successful operation.
400	Wrong request, missing parameters, missing header
404	Indicates property was not found
500	Indicate an internal service error

GET /customisations

Returns the list of existing customisations as a JSON array of customisation names.

(1) Request header

required	name	value	description
optional	Accept	application/json	Set the response type of the output

(2) Example response

Here a sample of the response:

```
[
  "ww1",
  "ww2",
  "biology"
]
```

GET /customisation/{name}

Retrieve the content of a specific customisation

(1) Parameters

required	name	content-type value	description
required	name	String	name of the customisation to be retrieved

(2) Request header

required	name	value	description
optional	Accept	application/json	Set the response type of the output

(3) Example response

Here a sample of the response

```
{
  "wikipedia": [
    4764461,
    51499,
    1014346
  ],
  "language": {
```

(continues on next page)

(continued from previous page)

```

    "lang": "en"
  },
  "texts": [
    "World War I (WWI or WW1 or World War One), also known as the First World War or_
↪the Great War, was a global war centred in Europe that began on 28 July 1914 and_
↪lasted until 11 November 1918."
  ],
  "description": "Customisation for World War 1 domain"
}

```

Or in case of issues:

```

{
  "ok": "false",
  "message": "The customisation already exists."
}

```

POST /customisations

Creates a customisation as defined in the input JSON, named following the path parameter. The JSON profile specifies a context via the combination of a list of Wikipedia article IDs and text fragments. A text describing informally the customisation can be added optionally.

If the customisation already exists an error is returned.

(1) Parameters

required	name	content-type value	description
required	name	String	name of the customisation to be created
required	value	String	JSON representation of the customisation (see example)

(2) Request header

required	name	value	description
optional	Accept	application/json	Set the response type of the output

(3) Example response

Here a sample of the response

```

{
  "ok": "true"
}

```

Or in case of issues:

```

{
  "ok": "false",
  "message": "The customisation already exists."
}

```

PUT /customisation/{profile}

Update an existing customisation as defined in the input JSON, named following the path parameter. The JSON profile specifies a context via the combination of a list of Wikipedia article IDs, FreeBase entity mid and text fragments.

A text describing informally the customisation can be added optionally.

(1) Parameters

required	name	content-type value	description
required	profile	String	name of the customisation to be updated

(2) Request header

required	name	value	description
optional	Accept	application/json	Set the response type of the output

(3) Example response

Here a sample of the response

```
{
  "ok": "true"
}
```

Or in case of issues:

```
{
  "ok": "false",
  "message": "The customisation already exists."
}
```

DELETE /customisation/{profile}**(1) Parameters**

required	name	content-type value	description
required	profile	String	name of the customisation to be deleted

(2) Request header

required	name	value	description
optional	Accept	application/json	Set the response type of the output

(3) Example response

Here a sample of the response

```
{
  "ok": "true"
}
```

Or in case of issues:


```
{  
  "ok": "false",  
  "message": "The customisation already exists."  
}
```


5.1 Datasets for long texts

It is possible to evaluate *entity-fishing* entity disambiguation models with several well-known available datasets. For convenience, the following datasets are present in the *entity-fishing* distribution:

- ``ace``: this is a subset of the documents used in the ACE 2004 Coreference documents with 36 articles and 256 mentions, annotated through crowdsourcing, see [1].
- ``aida``: AIDA-CONLL is a manually annotated dataset based on the CoNLL 2003 dataset, with 13881 Reuters news articles and 27817 mentions, see [2]. Note that the raw texts of this dataset are not included in *entity-fishing*, they have to be obtained from NIST (free for research purpose). AIDA-CONLL dataset can be considered as the most significant gold data for entity disambiguation both in term of size, ambiguity rate and annotation quality. In addition to the complete AIDA-CONLL dataset, this corpus is divided into three subsets that can be used for evaluation separately:
 - ``aida-train``: corresponds to the training subset of the CoNLL 2003 dataset
 - ``aida-testa``: corresponds to the validation subset of the CoNLL 2003 dataset
 - ``aida-testb``: corresponds to the test subset of the CoNLL 2003 dataset
- ``aquaint``: this dataset has been created by Milne and Witten [3], with 50 documents and 727 mentions from a news corpus from the Xinhua News Service, the New York Times, and the Associated Press.
- ``iitb``: manually created dataset by [4] with 50 documents collected from online news sources.
- ``msnbc``: this dataset is based on 20 news articles from 10 different topics (two articles per topic) and contains a total of 656 mentions, see [5].
- ``clueweb``: WNED-Clueweb 12 dataset is a large dataset created by [6] from the Clueweb corpora automatically - it is this far less reliable than the previous ones.
- ``wikipedia``: similarly as the Clueweb dataset, this set has been created automatically by [6] from Wikipedia, thus also clearly less reliable.
- ``hirmeos``: manually created dataset using open accessible books (licence CC-BY), financed from the European project H2020 Hirmeos [7].

All these reference datasets are located under *data/corpus/corpus-long*.

5.2 Evaluation commands

Use the following maven command with the above dataset identifier for running an evaluation:

```
$ ./gradlew evaluation -Pcorpus=[dataset]
```

For instance for evaluating against the testb subset of the AIDA-CONLL, use:

```
$ ./gradlew evaluation -Pcorpus=aida-testb
```

The evaluation process will provide standard metrics (accuracy, precision, recall, f1) for micro- and macro-averages for the entity disambiguation algorithm selected as ranker and for priors (as baseline).

The recall of the candidate selection with respect to the gold annotations is also provided (e.g. the proportion of candidate sets containing the expected answer before the ranking).

5.3 Generation of pre-annotated training/evaluation data

In case a new corpus needs to be created, *entity-fishing* includes the possibility to automatically generate an XML file of entity annotations from text or pdf files in the same format as the other existing corpus. These generated files can then be corrected manually and used as gold training or evaluation data, or they can be used for semi-supervised training.

For a given new corpus to be created, for instance the corpus *toto*, the following directory must be created: *data/corpus/corpus-long/toto/*. The documents part of this corpus must be placed under the subdirectories *RawText* and/or *pdf*.

If there is a directory called *pdf* or *PDF*, the process will extract information (title, abstract, body) from each pdf and save it as *pdfFileName.lang.txt* inside the *RawText* directory. The tool will then look into the subdirectory *RawText* and process the files **.txt* found inside. If the files name is in the form *filename.lang.txt* then the *lang* will be used as reference, otherwise *en* will be the default choice.

Use the following maven command with the above dataset identifier for generating the annotation xml file:

```
$ ./gradlew annotatedDataGeneration -Pcorpus=[corpusname]
```

For instance, for a new corpus *toto*, with text or pdf documents prepared as indicated above:

```
$ ./gradlew annotatedDataGeneration -Pcorpus=toto
```

5.4 References

[1] Lev-Arie Ratnov, Dan Roth, Doug Downey, and Mike Anderson. Local and global algorithms for disambiguation to wikipedia. In Dekang Lin, Yuji Matsumoto, and Rada Mihalcea, editors, The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA, pages 1375–1384. ACL. <<http://www.aclweb.org/anthology/P11-1138>>.

[2] Johannes Hoffart, Mohamed Amir Yosef, Ilaria Bordino, Hagen Fürstenau, Manfred Pinkal, Marc Spaniol, Bilyana Taneva, Stefan Thater, and Gerhard Weikum. Robust disambiguation of named entities in text. In Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, EMNLP 2011, 27-31 July 2011, John

McIntyre Conference Centre, Edinburgh, UK, A meeting of SIGDAT, a Special Interest Group of the ACL, pages 782–792. ACL. <<http://www.aclweb.org/anthology/D11-1072>>.

[3] David N. Milne and Ian H. Witten. Learning to link with wikipedia. In James G. Shanahan, Sihem Amer-Yahia, Ioana Manolescu, Yi Zhang, David A. Evans, Aleksander Kolcz, Key-Sun Choi, and Abdur Chowdhury, editors, Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM 2008, Napa Valley, alifornia, USA, October 26-30, 2008, pages 509–518. ACM. DOI <<https://doi.org/10.1145/1458082.1458150>>.

[4] Sayali Kulkarni, Amit Singh, Ganesh Ramakrishnan, and Soumen Chakrabarti. Collective annotation of Wikipedia entities in web text. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD ‘09), Paris, France, 2009, pages 457-466. ACM. DOI: <<https://doi.org/10.1145/1557019.1557073>>

[5] Silviu Cucerzan. Large-scale named entity disambiguation based on Wikipedia data. In Jason Eisner, editor, EMNLP-CoNLL 2007, Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, June 28-30, 2007, Prague, Czech Republic, pages 708–716. ACL. <<http://www.aclweb.org/anthology/D07-1074>>.

[6] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In Zoubin Ghahramani, editor, Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007, volume 227 of ACM International Conference Proceeding Series, pages 129–136. ACM. DOI <<https://doi.org/10.1145/1273496.1273513>>.

[7] HIRMEOS H2020 project. More information [here](#).

Train and evaluate

Trained models for entity recognition and disambiguation are provided in the project repository. The following section explains how to retrain the models.

6.1 Training with Wikipedia

Currently a random sample of Wikipedia articles is used for training. The full article content is therefore necessary and a dedicated database will be created the first time the training is launched. This additional database is used and is required only for training. You will need the Wikipedia XML dump corresponding to the target languages available in a directory indicated in the `yaml` config files by the parameter `dataDirectory`. A warning here, as this additional database contains the whole textual content of all Wikipedia articles (with wiki markups), it is quite big, around 7.6G GB for the English Wikipedia (dump from May 2020). This database (stored under the `dbDirectory` indicated in the language config file and called `markupFull`) will be built automatically if not present, so typically at first launch of the training for a given language, and the process will take a bit more than one hour for building the English version for example.

The following command will build the two models used in *entity-fishing*, the `ranker` and the `selector` model (Gradient Tree Boosting for the first one, Random Forest for the second one) and preliminary build the full article content database the first time for the English Wikipedia:

```
$ ./gradlew train_wikipedia -Plang=en
```

For other languages, replace the ending language code (`en`) by the desired one (`fr`, `de`, `it`, `es`, `ar`, `zh`, `ru` and `ja` are supported), e.g.:

```
$ ./gradlew train_annotate -Plang=fr
$ ./gradlew train_annotate -Plang=de
```

Models will be saved under `data/models`. ARFF training data files used to build the model are saved under `data/wikipedia/training/`.

6.2 Evaluation with Wikipedia

An evaluation is produced at the end of training base on a random sample of Wikipedia articles, providing macro- and micro-average precision, recall and f1-score.

Note that the ratio of disambiguated mentions in a Wikipedia article is low. As a consequence, the precision of our models will be very low because they are built for disambiguating a maximum of entities. Recall is probably a more meaningful measure when evaluating with Wikipedia.

For an evaluation of the NED aspect (ranker in our framework) with well-known datasets, which is much more standard and allows comparison with other similar works, see the evaluation section.

6.3 Training with an annotated corpus

It is possible to train the entity-fishing models with several well-known available datasets. For convenience, the datasets indicated here *Evaluation* are present in the *entity-fishing* distribution.

Use the following command with a dataset name and a language identifier for running a training with this dataset:

```
$ ./gradlew train_corpus -Pcorpus=aquaint -Plang=en
```

For instance for training with the train subset of the AIDA-CONLL, use:

```
$ ./gradlew train_corpus -Pcorpus=aida-train -Plang=en
```

entity-fishing also included the possibility to generate additional pre-annotated corpus, for instance to be further corrected manually. See *Evaluation* for the explanations.

The evaluation with annotated corpus is also described in the page *Evaluation*.

6.4 Creating entity embeddings

Entity embeddings are used to improve entity disambiguation. They are created from word embeddings and entity descriptions generated from Wikidata and Wikipedia. Embeddings resources are provided with the project data resources, so you normally don't have to create yourself these embeddings. For reference, we document here how to create these entity embeddings. The process is as follow:

1. Download available pretrained word embeddings for a target language - this could be for instance word2vec, FastText, or lexvec. Word embeddings need initially to be in the standard .vec format (a text format). word2vec binary format can be transformed into .vec format with the simple utility *convertvec*

Note: English and Arabic word embeddings used in the current *entity-fishing* are Glove “flavor”. Arabic embeddings are available at https://archive.org/details/arabic_corpus, see https://ia803100.us.archive.org/4/items/arabic_corpus/vectors.txt.xz. Other languages are using fastText word embeddings.

2. Quantize word embeddings

Quantize will simplify the vector given an acceptable quantization factor (by default the error rate for quantizing is 0.01, but it could be changed with the argument *-Perror*)

```
$ ./gradlew quantize_word_embeddings -Pi=/media/lopez/data/embeddings/glove-vectors.  
→vec -Po=/media/lopez/data/embeddings/word.embeddings.quantized
```


Here some Glove word embeddings `glove-vectors.vec` given as input (`-i`) will be quantized and saved as `word.embeddings.quantized`. By default, the flag `-hashheader` is used and indicates that the first line (a header to be ignored) must be skipped. In case there is no header, `-hashheader` should be removed in the corresponding gradle task `quantize_word_embeddings` (see file `build.gradle`).

3. Create Wikidata entity description to be used for producing entity embeddings. The command for creating description is the following one:

```
$ ./gradlew generate_entity_description -Plang=en
```

Replace the `en` argument by the language of interest.

The generated description are saved under `data/embeddings/en/`, given the language of interest (here `en`).

4. Create entity embeddings from the generated description.

This step might take a lot of time and exploiting multithreading is particularly helpful. The number of threads to be used is given by the argument `-n`:

```
$ ./gradlew generate_entity_embeddings -Pin=entity.description -Pv=word.embeddings.quantized -Pout=entity.embeddings.vec -Pn=10
```

The following parameters are available:

- **-h**: displays help
- **-in**: path to an entity description data file
- **-v**: the path to the word embedding file in `.vec` format (e.g. one originally of `word2vec`, `faster`, `lexvec`, etc.), optionally quantized
- **-out**: path to the result entity embeddings file (not quantized, this is to be done afterwards)
- **-n**: number of threads to be used, default is 1 but it is advice to used as much as possible
- **-rho**: rho negative sampling parameters, if it's < 0 use even sampling, default is -1 (must be an integer)
- **-max**: maximum words per entity, if < 0 use all the words, default is -1 (must be an integer)

5. Quantize entity embeddings

Finally, similarly as the steps 2., we apply a quantization to the entity embeddings:

```
$ ./gradlew quantize_word_embeddings -Pi=/media/lopez/data/embeddings/entity.embeddings.vec -Po=/media/lopez/data/embeddings/entity.embeddings.quantized
```

The entity embeddings are now ready to be loaded in the embedded database of *entity-fishing*.

6. Copy the quantized embeddings files (e.g. `entity.embeddings.quantized`) under the *entity-fishing* data repository (the one containing the csv files). *entity-fishing* expects compressed files with `.gz` extension: `word.embeddings.quantized.gz` and `entity.embeddings.quantized.gz`. Starting *entity-fishing* will load automatically the embeddings in the embedded database LMDB as binary data.

CHAPTER 7

License and contact

entity-fishing is distributed under [Apache 2.0 license](#). The dependencies used in the project are either themselves also distributed under Apache 2.0 license or distributed under a compatible license.

The documentation is distributed under [CC-0](#) license and the annotated data under [CC-BY](#) license.

If you contribute to entity-fishing, you agree to share your contribution following these licenses.

Main author and contact: Patrice Lopez (<patrice.lopez@science-miner.com>)