KCONNECT

www.kconnect.eu

Semantic annotation toolkit (first version)

Deliverable number	*D1.1*
Dissemination level	*Public*
Delivery date	*31 October 2015*
Status	*Final*
Author(s)	*Ian Roberts, Fredrik Axelsson, Zoltan Varju, Ljiljana Dolamic, Célia Boyer, Angus Roberts*

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644753 (KConnect)

Executive Summary

This report accompanies software, the KConnect Semantic Annotation Toolkit, and gives a description of, and instructions for the use of, that software. The toolkit is used to convert an existing semantic annotation application, the Khresmoi application, to other languages.

The Khresmoi project created an application for the semantic annotation of English and Czech health information texts. By semantic annotation, we mean the linking of health-related terms in the text, to concepts in some external knowledge base, in this case, the KConnect knowledge base. In creating the KConnect Semantic Annotation Toolkit, we have split the original Khresmoi pipeline in to its component parts, and identified those parts that are language dependent, and those that are language independent. The toolkit comprises templates and scripts for the adaptation of those parts to a new target language. The parts are:

- Basic processing (sentence splitting, part of speech tagging etc.)
- Stop-word handling
- Terminology lookup
- Terminology disambiguation

The toolkit has been applied to a test case language – French – and a French semantic annotation pipeline built using the toolkit. This is reported as an example of using the toolkit, along with an evaluation of the resulting toolkit.

Work is on-going to adapt the toolkit to other languages, and the availability of resources and components that could be useful has been surveyed for two of these languages, Hungarian and Swedish. Suitable components have been identified, and adaptation for the toolkit has been started.

Version two of the toolkit will provide more sophisticated disambiguation, based on word frequency and supervised language models; tools for the adaptation of these models to other languages; and further adaptation of the complete application to the target languages.

Table of Contents

List of Abbreviations

POS	Part-of-speech, e.g. verb, noun, adjective
NER	Named Entity Recognition
MeSH	Medical Subject Headings, a taxonomy used to index medical literature
NLP	Natural Language Processing
UMLS	Unified Medical Language System, a large terminology resource for biomedicine
CUI	Concept Unique Identifier, a unique identifier for concepts in the UMLS
GATE	General Architecture for Text Engineering, a text analytics toolkit
XML	eXtensible Markup Language, a machine-readable syntax for describing data
JSON	JavaScript Object Notation, a data interchange format
ISO 639	International Standards Organisation standard 639, codes for specifying languages
URI	Uniform Resource Identifier, a unique identifier for a resource
TUI	Type Unique Identifier, a unique identifier for semantic types in the UMLS
HON	Health On the Net
CLEF	Cross-Language Evaluation Forum
EMEA	European Medicines Agency
BRAT	Rapid annotation tool

1 Introduction

The Khresmoi project developed a semantic annotation system for biomedical literature and other medical texts in the English language. The system consisted of several components, some language-independent and some English-specific. One goal of KConnect is to provide a toolkit to simplify adaptation of the Khresmoi system to languages other than English. This report accompanies the first version of the toolkit, and provides documentation and instructions for the toolkit, together with a description of a French test case and ongoing adaptations to other languages.

1.1 The Semantic Annotation Toolkit

The toolkit accompanying this report comprises two files:

- adaptation-tools.tgz – the semantic annotation toolkit
- french-pipeline.tgz – an example French pipeline built using the toolkit

Both consist of tar'd and gzip'd directories. To use the toolkit, un-tar and un-gzip in a convenient location, and follow the instructions in this report. To use the French pipeline, un-tar and un-gzip in a convenient location, and load as a GATE application[1] in either GATE Developer, or any application using GATE Embedded, or in a framework capable of distributing a GATE application (such as the KConnect cloud service).

1.2 Structure of this report

Section 2 describes the Khresmoi semantic annotation pipeline in detail, breaking it down into its various language dependent and independent components. Section 3 gives detailed instructions for the work required to adapt each component to a new language. Section 4 acts as an illustration of the previous section, describing a test case adaptation to French, together with an evaluation. Sections 5 and 6 describe initial work on adaptation to Swedish and Hungarian, and Section 7 describes future work planned for version 2 of the toolkit.

1 https://gate.ac.uk

2 The Khresmoi semantic annotation pipeline

The Khresmoi semantic annotation pipeline for English language biomedical text was designed in a modular fashion, and consists of the following modules:

1. Basic structural and morphological annotations – tokenisation, sentence splitting, part-of-speech (POS) tagging and morphological analysis
2. Boilerplate detection – determining which sections of the document constitute the significant content and which are boilerplate such as page headers, footers, sidebars, etc.
3. Stopword detection – marking stopwords which should not be considered part of a term
4. Dictionary lookup against knowledge sources from the Khresmoi knowledge base – for English this is the UMLS and DrugBank
5. Heuristic disambiguation rules for ambiguous terms, for example given overlapping term candidates keep the longest

Some modules are more language-specific than others:

- The GATE tokeniser works reasonably well for most Western languages that have a similar structure to English in terms of white space separated tokens and standard sentence punctuation (full stops, question marks, etc.). However the tokeniser interacts closely with the POS tagger.
- The English pipeline makes use of POS tagging and morphological analysis in order to make the dictionary lookup phase more general – the lookup module looks for both exact matches between the text and the dictionary and matches based on the morphological roots. The POS tagger and morphological analyser are clearly English-specific, though GATE also includes a simpler stemmer which has support for many more languages.
- The stopword list is a language-specific dictionary.
- The terminology dictionaries are language specific, though given the predominance of English in the scientific world the English-language dictionaries may help boost performance even on non-English text.
- The heuristic rules will need to be tested for each new language.

3 Adapting the pipeline to new languages

The process of adapting the Khresmoi pipeline to a new language has a series of steps, many of which are optional depending on the resources available for the target language. The principal question is whether there is a POS tagger and/or stemmer or morphological analysis tool available in the desired language. In the absence of these, only basic exact-match dictionary lookup is possible. If a POS tagger is available then the heuristics can take part of speech information into account. If a morphological analyser is available then the dictionary lookup stage can match by morphological root as well as by exact string match.

The adaptation toolkit consists of a set of template GATE applications for the various sub-pipelines that make up the overall Khresmoi pipeline, along with some tools to convert data files into the appropriate formats and assemble the components to produce a new pipeline for the target language. These tools are described in the following sections, along with the more specific adaptation processes that cannot be supported by simple tools. To run the adaptation tools you will require:

- Java JDK 7 or later
- An installed copy of GATE 8.1

3.1 Tokeniser, POS tagger, morphological analysis

These three components closely interact. For many western languages the standard GATE Unicode tokeniser will produce sensible results, but if you wish to use some other POS tagger or morphological analyser then the tokenisation may need to be adjusted to match that expected by the tagger. In English, for example, the Hepple POS tagger (HepTag) expects "don't" to be two tokens, "do" and "n't" where the plain Unicode tokeniser creates three, "don", apostrophe, and "t", so the English tokeniser has a post-processing step to correct this.

GATE includes support for several different POS taggers by default:

- GATE's own native HepTag tagger (the Hepple PPOS tagger)
- The OpenNLP maximum entropy POS tagger[2]
- The Stanford log-linear POS tagger[3]

The latter two taggers offer pre-trained models for a number of different languages, as well as training tools to allow you to create your own model from a manually-tagged training corpus.

For morphology, GATE includes a native morphological analyser for English and also support for the Snowball stemmer[4], providing basic stemming in 16 different languages.

Since this sub-pipeline is potentially tightly customised to the target language it is difficult for us to provide tool support for adaptation. Several sample (English) pipelines are provided under the "basic-annots" folder, for the target language you will need to create your own pipeline in GATE Developer (possibly using one of these as a starting point). If one or other of the components described above are not suitable, then a suitable equivalent for the target language must be found, and wrapped for the GATE API, as described in the GATE user guide[1]. The final pipeline must generate the following annotations and features:

2 http://opennlp.apache.org

3 http://nlp.stanford.edu/software/tagger.shtml

4 http://snowball.tartarus.org

- Token: individual words
  - string: the plain string of the token
  - kind: word, number, symbol or punctuation
  - orth: lowercase, upperInitial (first letter uppercase, the rest lowercase), allCaps (all uppercase) or mixedCaps (any other mixture of upper and lower case)
  - category: part-of-speech tag (optional)
  - root: morphological root (optional)
  - stem: stem (optional)
- SpaceToken: spaces between words
- Sentence: sententences, including the terminating punctuation if any

Other annotations such as Split (for the sentence boundaries) as created by the default GATE components are acceptable but not strictly required by downstream components. It is good practice to add a "document reset PR" to the front of the pipeline but this should be configured so it only deletes the annotations that this pipeline created and not other annotations created by other components.

Any custom plugins that are not provided with the default GATE distribution should be placed in the top level "plugins" directory in the toolkit template structure.

3.2 Stopword list

The English pipeline includes a "stopword" list of words that should be ignored when doing dictionary lookups. Typically these are single letter words, closed class words such as prepositions, conjunctions, etc. If a similar list is available for the target language it can be used in the same way.

Stopwords will typically be matched using exact string matching, but if a morphological analyser is included in the basic annotations sub-pipeline then you may additionally wish to match stopwords based on their morphological roots. Whether this approach is sensible depends on the characteristics of the target language.

A tool is provided to take a stopword list and convert it into the form required for the Khresmoi pipeline. The tool is in the form of a command line script which takes its configuration from an XML file. In the "annotate-stopwords" folder of the adaptation toolkit, you will find an example configuration file "config-sample.xml" – make a copy of this file as "config.xml" and edit it appropriately. The configurable options are:

- language: the ISO 639 two letter language code for the target language, "en" for English, "fr" for French, etc.
- input: the path to the stopword list file, e.g. "stopwords.txt"
- basicApplication: the path to the "basic annotations" GATE saved application you created in the previous step, which will perform the tokenisation and optionally POS tagging and morphological analysis
- wordAnnotationType: the annotation type representing words, which should be left as the default value "Token" unless you are sure you need to change it
- spaceAnnotationType: the annotation type representing spaces between words, which should be left as the default value "SpaceToken" unless you are sure you need to change it

The remaining "feature" elements should name the annotation features you want to use for matching. Typically the first one will be "string", which denotes exact string matching. If you are using a stemmer or morphological analyser then you should add "root" or "stem" as appropriate.

To run the tool, set your GATE_HOME and JAVA_HOME environment variables to the locations where you have installed GATE Developer (8.1) and Java JDK (7 or later), and run the `stopwordsApp.sh` script. This will generate a number of new files in a "resources" folder, and a top level `application.xgapp` file containing a saved GATE application that will match stopwords based on all the features you specified in the configuration.

3.3 Terminology dictionaries

The core of the pipeline is the dictionary mapping expressions likely to be found in the text on to URIs in the KConnect knowledge base, principally the UMLS and DrugBank. The pipeline represents matched terms as GATE annotations, with features "inst" and "class" holding the relevant URIs in the KConnect knowledge base[5].

- For UMLS[6] terms, the "inst" is http://linkedlifedata.com/resource/umls/id/<CUI> and the "class" is http://linkedlifedata.com/resource/semanticnetwork/id/<TUI>
- For DrugBank[7] terms the "inst" is http://linkedlifedata.com/resource/drugbank/drug/<id> and the "class" is http://linkedlifedata.com/resource/drugbank/Drug

The knowledge base also contains terms from RadLex[8], but here the mapping is more complex as the class hierarchy of the RadLex ontology is much finer grained. The English pipeline does not use RadLex terms directly, but as many RadLex terms are cross-referenced to their UMLS equivalents they could be used as an alternative source of lexicalisations for these UMLS CUIs.

In order to adapt the Khresmoi pipeline to a new language, you must provide one or more *dictionary files* containing terms in the target language. Tools are provided to convert these dictionary files into the format required by the pipeline. Each line in the dictionary file should contain fields separated by a tab character (U+0009):

- For UMLS-linked dictionaries the row should have three fields, the lexical form, the CUI (C#######), and the TUI (T###)
- For DrugBank-linked dictionaries the row should have two fields, the lexical form and the DrugBank identifier

A pair of tools, one for UMLS and one for DrugBank, are provided to convert the dictionary files into the appropriate format for the Khresmoi pipeline. The tools are found in the "umls" and "drugbank" folders of the adaptation toolkit, and as with the stopwords tool they take their configuration from an XML file named "config.xml" in the respective directory. Sample configuration files are provided with comments explaining the available options, the common options are as follows:

- language: the ISO 639 two letter language code for the target language, as in the previous section
- basicApplication: the path to the "basic annotations" GATE saved application you created earlier, which will perform the tokenisation and optionally POS tagging and morphological analysis
- stopwordsApplication: the path to the "annotate-stopwords GATE saved application you created in the previous step, which will detect stopwords within the dictionary entries
- wordAnnotationType: the annotation type representing words, which should be left as the default value "Token" unless you are sure you need to change it
- spaceAnnotationType: the annotation type representing spaces between words, which should be left as the default value "SpaceToken" unless you are sure you need to change it

5 http://linkedlifedata.com

6 http://www.nlm.nih.gov/research/umls/

7 http://www.drugbank.ca/

8 http://www.radlex.org/

The configuration should specify one "dictionary" element per dictionary file:

```
<dictionary file="filename" alias="alias"
             caseSensitive="true|false">
  <feature />
  <feature>root</feature>
</dictionary>
```

The "file" attribute is the path to the tab separated dictionary file as described above, the "alias" should be a short descriptive name which will become part of various file names generated as part of the adaptation process, and "caseSensitive" specifies whether matching should be exact (true) or case-insensitive (false). The dictionary element has one or more child "feature" elements specifying the features of the word annotations that should be used to perform the matching. An empty "feature" element (the first example above) denotes an exact match based on the surface string forms in the text. Other features would typically be those generated by the stemmer or morphological analyser of the basic annotations pipeline.

For UMLS dictionary files there are two additional optional settings. The "wordFeatures" element specifies heuristic rules to determine which words are allowed to be the start and/or end of a match in the text, based on features of the word annotations. These rules would commonly be based on part of speech tags, e.g. "matches may only start with a noun or an adjective". The rules are specified as child elements named "start" or "end" under "wordFeatures", e.g.

```
<wordFeatures>
  <start feature="category" regex="N.*|JJ.*|VBD|VBN|VBG" />
  <end feature="category" regex="N.*" />
</wordFeatures>
```

The "feature" is the word feature being tested, the "regex" is a regular expression which the feature value must match in order to be permitted as the start (or end, as appropriate) of a match. The regex is tested against the *whole* of the feature value, so you must add ".*" for a prefix match. The example rules above allow matches to start on any kind of noun or adjective, or one of three specific subtype of verb, and to finish only on a noun. If there are *no* "start" (respectively "end") rules specified then all words are considered as valid locations for the start (respectively end) of a match.

The other option available for UMLS dictionaries is a "veto" list, specified as one or more "veto" elements:

```
<veto file="veto-investigation.txt" type="Investigation" />
```

The "file" is a simple list of terms, one per line, and the "type" specifies the annotation type for which this is a veto list (Disease, Investigation, Anatomy, Drug). Any match of a dictionary file which is also a match in the veto list for the appropriate annotation type will be ignored. This facility can be used to make exceptions to the main dictionaries (e.g. if they are automatically generated and you want to exclude certain common phrases without modifying the dictionaries themselves), or to mitigate against over-generation when matching based on morphological roots.

The DrugBank sub pipeline is less sophisticated than the UMLS one, and does not support word feature heuristics or veto lists.

To run the tools, set your GATE_HOME and JAVA_HOME environment variables to the locations where you have installed GATE Developer (8.1) and Java JDK (7 or later), and run the `umlsApp.sh` or `drugBankApp.sh` script as appropriate. This will generate a number of new files in the "resources" and "gazetteer" folders, and a top level `application.xgapp` file containing a saved GATE application that will match stopwords based on all the features you specified in the configuration.

3.4 Disambiguation rules

An example set of heuristic disambiguation rules are provided in the "disambiguate" folder of the adaptation toolkit. The rules are implemented as a simple multi-phase JAPE grammar, so there is no need for a tool to generate a customised pipeline, you can simply edit the "main.jape" file to include or exclude phases as appropriate for the target language, or to add your own phases by writing your own JAPE rules. The default phases are:

- longest-only.jape: at any given location, keep only the longest UmlsLookup annotations that start at that location
- remove-dups.jape: remove any duplicate UmlsLookup annotations, i.e. where there is more than one annotation covering the same span and marked with the same CUI, delete all but one of them
- highest-cui.jape: given a set of annotations covering the same span, keep just the single annotation with the highest numbered CUI, and delete the rest
- clean-overlaps.jape: remove any remaining partially-overlapping UmlsLookup annotations.

3.5 Assembling the complete application

Once you have adapted the various sub-pipelines the final step in the adaptation process is to assemble the "master" application that calls all the sub pipelines in the correct sequence and generates the final output annotations. To do this, use the `assembleApp.sh` script located in the top level directory of the adaptation toolkit. This script expects the list of sub pipeline files as command line options, and will generate a top-level `application.xgapp` for the master application. An example invocation would be (all on one line):

```
./assembleApp.sh   basic-annots/basic-annots-en.gapp
                   annotate-stopwords/application.xgapp
                   umls/application.xgapp
                   drugbank/application.xgapp
                   disambiguate/application.xgapp
```

The resulting application is now ready to be loaded into GATE or used with other GATE tools for testing on the sample data you prepared earlier.

3.6 Test data

In order to evaluate and tune the performance of the pipeline it is extremely useful to have some manually-annotated data available for testing purposes. Ideally the data should be annotated with entities linked to UMLS CUIs, but even basic annotations giving just the entity type (without a CUI link) will be valuable.

If test data can be prepared as GATE documents then you will be able to use the GATE corpus quality assurance tools to evaluate your adapted pipeline's performance. Alternatively, GATE allows you to export annotated documents as XML or JSON, for evaluation using other tools.

4 Case study: adapting the pipeline for French

As a test run of the adaptation toolkit, this section walks through the process of adapting the Khresmoi pipeline to the French language, using terminology resources and test data made available by KConnect project partner Health on the Net (HON).

4.1 Tokeniser, POS tagger, morphological analysis

For French, we were able to use the French tokeniser provided with GATE, and the default French POS tagger model for the Stanford tagger. For morphological analysis we tried the Snowball stemmer provided with GATE, and also the Morfette tool[9], for which a custom GATE wrapper needed to be written.

4.2 Stopword list

Two different French stopword lists were provided by HON, one containing 126 words as used by HON during their indexing of French documents, and the other a more extensive list of 463 words generated by a previous research project.

4.3 Terminology dictionaries and heuristics

The terminology dictionary for French was constructed by extracting French labels directly from the Khresmoi knowledge base. These labels were extracted along with their corresponding UMLS CUIs and TUIs, in the tab separated format required by the adaptation toolkit. The default heuristic rules were used, so removing overlapping matches and preferring the match with the highest CUI in the case of multiple annotations with the same span.

4.4 Test data

For test data, we use a subset of the QUAERO French Medical Corpus[1] used for the CLEF eHealth 2015 shared task[10], which consists of titles of MEDLINE articles and also longer documents containing information about drugs from the European Medicines Agency (EMEA), all annotated with named entities linked to UMLS CUIs. The QUAERO corpus includes annotations covering a wider range of UMLS semantic types than the Khresmoi pipeline is designed to find, so for testing purposes we restricted to just annotations of the 30 semantic types targeted by the pipeline. Out of the 5,690 individual annotated mentions in the data we used (comprising 2,306 distinct CUIs), 3,363 mentions (1,347 distinct CUIs) belonged to one of the 30 target semantic types.

The data is provided in the BRAT standoff annotation format, and was converted into GATE documents using GATE's BRAT format reader.

9 https://hackage.haskell.org/package/morfette

10 https://sites.google.com/site/clefehealth2015/task-1/task-1b

4.5 Evaluation

With all the elements in place, the adaptation toolkit was used to produce several different versions of the French pipeline with different combinations of the snowball stemmer or Morfette morphological analyser, and one of the two stopword lists. The four variants of the pipeline were then tested against the QUAERO test data assembled above, using the GATE QA tools, as describes in the GATE user guide[1]. The results shown are "average" precision, recall and F1, where a partial match is given half the weight of an exact match.

This first table gives the "headline" results including all four annotation types (Anatomy, Disease, Drug and Investigation), where an annotation must have the correct type and the correct CUI to be considered a successful match:

Pipeline	Precision	Recall	F1
Morfette / 463 stopwords	67.87	23.10	34.46
Morfette / 126 stopwords	65.70	22.95	34.01
Stemmer / 463 stopwords	62.05	24.31	34.93
Stemmer / 126 stopwords	61.38	24.57	35.10

Precision is reasonable here – of the mentions that the pipeline found, 60-70% of them were correct. Recall is much lower, with the pipeline missing many annotations that are marked in the test data. Inspection of the data suggests two main reasons for this. First the heuristic rules prevent the pipeline from annotating overlapping mentions, whereas in the test data there are many instances of such overlaps, e.g. for the string "cancers pulmonaires" both "cancers" and "cancers pulmonaires" are annotated in the test data but the pipeline only reports the longer one. Second the pipeline lacks a good dictionary of drug terminology, and it also misses a number of very general "findings" (T033) that are annotated in the test data, such as "résultats". Nontheless, the purpose of this test was not so much to evaluate how well the pipeline performs overall, but rather which variant of the pipeline is most interesting for further development.

Breaking the results down by annotation type we see that some types show much better performance than others:

Anatomy	Precision	Recall	F1
Morfette / 463 stopwords	77.78	14.50	24.45
Morfette / 126 stopwords	79.07	14.09	23.92
Stemmer / 463 stopwords	66.87	15.33	24.94
Stemmer / 126 stopwords	67.07	15.19	24.77

Disease	Precision	Recall	F1
Morfette / 463 stopwords	69.02	37.05	48.22
Morfette / 126 stopwords	67.42	37.12	47.88
Stemmer / 463 stopwords	63.87	39.04	48.46
Stemmer / 126 stopwords	63.26	39.37	48.53

Drug	Precision	Recall	F1
Morfette / 463 stopwords	88.41	8.53	15.56
Morfette / 126 stopwords	86.76	8.25	15.07
Stemmer / 463 stopwords	83.78	8.67	15.72
Stemmer / 126 stopwords	83.54	9.23	16.62

Investigation	Precision	Recall	F1
Morfette / 463 stopwords	34.82	10.40	16.02
Morfette / 126 stopwords	29.69	10.13	15.11
Stemmer / 463 stopwords	31.34	11.20	16.50
Stemmer / 126 stopwords	30.07	11.47	16.60

Generally, the larger stopword list provides better performance, and Morfette gives higher precision but lower recall than the stemmer. It is also worth noting that the stemmer is significantly faster to run than the Morfette tagger, and this may be a consideration in high throughput situations.

5 Adapting components for the Swedish language

Investigation of existing Swedish tools, along with adaption of these resources to the GATE framework has started, and is described below.

5.1 Overview of components and resources available for Swedish

This is an overview of the language processing tools existing for Swedish, which can be used out-of-the-box. Some of them are already wrapped as GATE components in existing GATE plugins, others need to be wrapped.

A widely used machine learning based toolkit for text processing, the Apache OpenNLP toolkit[11], contains trained models for the Swedish language. Parts of the OpenNLP toolkit is already wrapped as GATE processing resource components using Maxent (maximum entropy) models: Named Entity Recognition, chunker, part-of-speech tagger, sentence splitter and tokeniser. Also a syntactic parser from OpenNLP has been wrapped[12]**.** For Swedish, trained models for tokeniser, sentence detector and POS-tagging (Maxent and perceptron) exists trained on a Swedish Treebank[13]. Another statistical part-of-speech tagging component exists using HunPos, trained on the balanced Swedish Stockholm Umeå Corpus (SUC)[14].

For stemming, a Swedish version of the popular Snowball stemmer is available[15]. This component is also wrapped as a GATE processing resource[16].

Findwise uses an internal lemmatizer based on a morphological lexicon from SALDO[17] for doing morphological analysis of tokens.

For additional morphological analysis, Findwise uses an internal compound word splitter based on SALDO, for splitting a word in several morphological components.

For dependency parsing, a data-driven system MaltParser[18] is used by Findwise. A trained model using the Swedish Treebank data exists[19].

Three main medical terminology dictionaries are used for Swedish:

- Swedish MeSH[20]**,** a taxonomy with medical subject headings.
- Swedish Snomed CT[21], a systemized nomenclature of Medicine and clinical terms.
- ICD-10[22] including codes for diagnosis.

11 https://opennlp.apache.org/

12 https://gate.ac.uk/sale/tao/splitch23.html#sec:misc-creole:opennlp

13 http://opennlp.sourceforge.net/models-1.5/

14 http://stp.lingfil.uu.se/~bea/resources/hunpos/

15 http://snowball.tartarus.org/algorithms/swedish/stemmer.html

16 https://gate.ac.uk/sale/tao/splitch23.html#sec:parsers:stemmer

17 http://spraakbanken.gu.se/swe/resurs/saldo

18 http://www.maltparser.org/

19 http://www.maltparser.org/mco/mco.html

20 http://mesh.kib.ki.se/swemesh/swemesh_en.cfm

21 https://www.socialstyrelsen.se/nationellehalsa/snomed-ct

22 https://www.socialstyrelsen.se/klassificeringochkoder/diagnoskodericd-10

Three main pieces of work for identifying medical terms in Swedish medical text have contributed to ideas for adapting the GATE pipeline for Swedish. Muntaga et al. [2] identify pharmaceutical drugs in clinical text, Skeppstedt et al. [3] used a rule-based model to identify disorders, findings and body structure in clinical text. Skeppstedt et al. [4] also identified the same types of terms in a study using a data-driven model.

5.2 Components adapted

The basic text analyzing pipeline for Swedish makes use of a tokenizer, sentence splitter, part-of-speech tagger, stemmer, lemmatizer, compound word splitter and entity recognition. Not all of these components are needed for all words, but they can complement each other when a previous component cannot extract its intended feature, i.e. lemmatizer and stemmer.

The GATE ANNIE tokenizer and GATE ANNIE sentence splitter for English performed as well as the Swedish components with initial testing of the pipeline. However, it may be good to use the same tokenizer with which the PoS-tagger model is trained with. Therefore, the openNLP tokenizer and sentence splitter components are used in current Swedish pipeline.

Components already existing in GATE

The Swedish basic language pipeline makes use of the following components, where some already are wrapped in GATE:

- Swedish sentence splitter in openNLP-plugin. (Already wrapped)
- Swedish Tokenizer in openNLP-plugin. (Already wrapped)
- Maxent PoS-tagger from openNLP-plugin with Swedish model. (Already wrapped)
- Swedish Snowball stemmer. (Already wrapped)
- Findwise Swedish lemmatizer
- Findwise Swedish Compound word splitter
- Basic medical named entity recognition based on gazetteers with dictionaries from Snomed CT SE, SweMeSH and ICD-10.

Currently no dependency parsing is done for Swedish since it is not used by the subsequent components.

Swedish lemmatizer

Based on a morphological lexicon this lemmatizer knows different forms of a word. Given a string representation of a word the components provide the lemma for the given word.

The KConnect lemmatizer component

This component was wrapped by extending GATE AbstractLanguageAnalyser and implementing GATE ProcessingResource. During inititalization of the component, the model is loaded. During execution the processing step iterates through each token in either the default or a specified annotation set. Each token get an extra feature named “lemma” with the lemma of the token string, produced by the lemmatizer.

Swedish compound word splitter

Based on a morphological lexicon this compound word splitter knows the different forms of a word which can start and end a compound word. Given a string representation of a word the components provide possible splitting’s of the given word, with the lemma for the separate parts of the word[23].

23 http://labdemos.findwise.com/cs-swe/cs-swe

The KConnect compound word splitter component

This component was wrapped by extending GATE AbstractLanguageAnalyser and implementing GATE ProcessingResource. During inititalization of the component the model is loaded. During execution the processing step iterates through each token in either the default or a specified annotation set. Each token get an extra feature named "compound" if it is a compound word. The feature contains a list of the lemmastized versions of separate words in the compound word, produced by the compound word splitter.

5.3 Next steps

To finalize the Swedish pipeline, the following components still need to be adapted:

- Negation component
- Abbreviation components
- Improved named entity recognition
- Named entity recognition of drugs for Swedish

What resources still need to be found?

A taxonomy containing information in Swedish regarding drugs, i.e. Farmacevtiska specialiteter i Sverige (FASS)[24]**,** needs to be incorporated in the knowledge base which provide the gazetteer lists.

24 http://www.fass.se/

6 Adapting components for the Hungarian language

This section provides a brief overview of tools and resources available for the Hungarian language.

6.1 Overview of components and resources available for Hungarian

Available software components

The Snowball-stemmer4 is widely used in enterprise search applications, and its Hungarian adaptation is available for various programming languages and platforms. Since stemming is the process of removing inflectional and derivational endings from words, applying this method to agglutinative languages often yields unnatural and bad results.

The most popular morphological analyser for Hungarian is hunmorph[25] and it is based on the MySpell and HunSpell spellchecker and analysis libraries[26]. hunpos[27] is a very effective part-of-speech-tagger that implements a variant of the TnT tagger[28] hunmorph and hunpos were implemented in Ocaml. Although they were high quality software at the time of their creation, they are no longer actively maintained.

The Trendminer Hungarian Processing Pipeline[29] is a suit of scripts which are extending the capabilities of hunmorph and hunpos to deal with the challenges of analysing linguistic data collected from various social media platforms. The toolkit contains Python wrappers for hunmorph and hunpos and helper functions to normalize data. It is not actively maintained.

magyarlanc[30] is a Java based toolkit which contains the following modules: Sentence splitter, Tokenizer, POS tagger, Lemmatizer, Dependency parser

Available linguistic resources

The following linguistic resources are accessible for Hungarian.

morphdb.hu[31] is a morphological database containing lemmas and production rules. The database is open with permissive licensing. Every above-mentioned tool is using a version of hunmorph for lexical analysis.

Szeged Treebank[32] is a high quality, balanced corpus for Hungarian. It contains various annotations, e.g. part-of-speech information, reference resolutions, and etc. It is very popular for training Hungarian NLP tools, every tool in the previous section uses a model based on the Szeged Treebank. Since it contains copyrighted materials, using it in enterprise environment requires licensing it from its creators.

25 http://mokk.bme.hu/resources/hunmorph/

26 http://hunspell.sourceforge.net/

27 https://code.google.com/p/hunpos/

28 http://www.coli.uni-saarland.de/~thorsten/tnt/

29 https://github.com/mmihaltz/trendminer-hunlp

30 http://www.inf.u-szeged.hu/rgai/magyarlanc

31 http://mokk.bme.hu/resources/morphdb.hu/

32 http://rgai.inf.u-szeged.hu/SzegedTreebank

Szeged NER Corpus[33] contains two collections of newswire texts. One is annotated for various business related named entities, one contains criminal news items along with annotations of crimes, victims and etc. The NER corpus derived from the Szeged Treebank corpus and can be used under the same restrictions.

huNERwiki[34] is corpus automatically generated from the Hungarian Wikipedia.

MeSH – Akadémiai Kiadó (Academic Press), the client of Precognox, translated the entries of MeSH and provided us with the permission to use it in KConnect services.

6.2 Components adapted

magyarlanc

magyarlanc provides tools for the basic NLP tasks, namely sentence splitting, tokenisation, lemmatisation, part-of-speech tagging and dependency parsing.

The Kconnect Magyarlanc component

The KConnect magyarlanc component is an adaptation of magyarlanc for GATE. It consists of the sentence splitter, the tokenizer and the POS-taggers wrapped as GATE components. Since no KConnect service plans to use the dependency parsing capabilities, it has not been built into the KConnect magyarlanc component. Each component extends AbstractLanguageAnalyzer and implements GateProcessingResource, both from the GATE API. String constants follow the naming conventions of ANNIE. Currently, magyarlanc is not a modularised library, and various parts of the software are being called simultaneously during execution.

6.3 Next steps

To finalize the Hungarian pipeline, the following task should be solved in the near future:

- modularisation of magyarlanc
- NER of persons, drugs and etc. for Hungarian

33 http://rgai.inf.u-szeged.hu/corpus_ne

34 http://hlt.sztaki.hu/resources/hunnerwiki.html

7 Conclusions and future work

We have created the first version of a toolkit to adapt the Khresmoi semantic annotation pipeline to languages other than English. This has been tested by creating a French pipeline, which was evaluated against an existing corpus. Although the toolkit was capable of creating a French pipeline from the resources available, performance of the pipeline was poor. This was not, however, a failing of the toolkit, but rather due to the fact that the only available evaluation corpus targets different concepts and terms to the Khresmoi pipeline.

Future work needs to address this, and attempt a more principled evaluation.

A second version of the toolkit is due for delivery in July 2016. It is planned to contain the following additions:

- Complete adaptation to Hungarian and Swedish
- Model based disambiguation, that can be trained for new languages
- Tools for training the above
- Tools for adapting to different document metadata requirements
- Tools for configuring indexes of annotations created by semantic annotation applications
- Further evaluation
- A short manual, based on this document, and on feedback in response to this document

8 References

[1] Névéol A, Grouin C, Leixa J, Rosset S, Zweigenbaum P. (2014) The QUAERO French Medical Corpus: A Ressource for Medical Entity Recognition and Normalization. Fourth Workshop on Building and Evaluating Ressources for Health and Biomedical Text Processing - BioTxtM2014. 24-30

[2] Muntaha S, Kvist M, Dalianis H. (2012) Enity Recognition of Pharmaceutical Drugs in Swedish Clinical Text. In proceedings of the Fourth Swedish Language Technology Conference. p. 77-78.

[3] Skeppstedt M, Kvist M, Dalianis H. (2012) Rule-based Entity Recognition and Coverage of SNOMED CT in Swedish Clinical Text. In proceedings of the Eight International Conference on Language Resources and Evaluation. p. 1250-1257.

[4] Skeppstedt M, Kvist M, Nilsson GH, Dalianis H. (2014) Automatic recognition of disorders, findings, pharmaceuticals and body structures from clinical text: An annotation and machine learning study. Journal of Biomedical Informatics 49. p. 148-158.