D6.3NE Software Package Guide

Robert Bossy

July 10, 2006

Contents

Part I

Installation

1 Requirements

The D6.3NE package is primarily developed for Linux but it should work (though not tested) on any Unix platform. It requires that the following software is installed on your system:

- to install the package:
  - GNU tar (1.15.1) and gzip (1.2.4) or bzip2 (1.0.2)
  - or zip (2.3)
- to run the tools:
  - Perl (5.8.6) and the following packages must be installed:
    - Getopt::Long (2.68)
    - XML::DOM (no version provided)
    - XML::DOM::XPath (1.11)
  - xsltproc (1.1.12)
- to benefit from the framework:
  - GNU Make (3.80)
  - GNU xargs 4.1.20
  - Java (1.4.2)
  - CADIXE (2.0a4)
  - Weka 3.4.7
- to recompile this guide:
  - pdflatex from the kpathsea (3.5.4)
  - fig2dev (3.2)

2 Installation

Get the archive in your favourite compression format, and decompress it by typing either:

- `tar -x -z -f D6.3NE.tar.gz`
- `tar -x -j -f D6.3NE.tar.bz2`
- `unzip D6.3NE.zip`

It will create a `D6.3NE` directory in your current working directory. This directory contains a `dtd` directory with the DTD definitions introduced by the package, a `tools` directory with all the scripts, a `corpus` directory that constitutes the main framework, and a `corpus_transcript` which is a copy of `corpus` filled with the biology example corpus.

There is no need to install it in the root directory, it is meant to work locally.

Part II

General work flow

The goal of the D6.3NE software package is the acquisition of named entities recognition (NER) resources by using machine learning. The process is split into five consecutive steps:

1. corpus selection;
2. automatic annotation of NEs;
3. manual validation of the annotation;
4. machine learning attributes selection;
5. learning.

Each step will produce modified version of the original corpus that will be used in the following step. In the rest of the section, we will describe each step, the expert knowledge required and the software provided to achieve them. The D6.3NE software package contains a suite of scripts and a framework directory tree to handle steps 3, 4 and 5. However we give some guidelines to get the steps 1 and 2 done.

3 Corpus selection

This phase depends essentially on what is available for the domain one wishes to acquire NEs. Standard corpora may already exist, but most of the time, you will have to build it by yourself.

We advise to select documents both rich in NE content and typical of the kind of documents you will apply NER. Indeed this corpus will be used as a training/testing set for machine learning algorithms. In other words, select web pages if you wish to extract NEs in web pages, select press releases if you wish to process press releases, select academic texts if you wish to process academic texts. If you want something more generic, the corpus should contain a balanced combination of the document types you want to process.

Having this corpus in Alvis Enriched Document format is an excellent idea, since it will allow you to use the Alvis NLP line for the following step.

4 Automatic annotation of NEs

The goal of this step is to have a broad linguistic annotation of the corpus gathered in step 1. The first use of this automatic annotation is a pre-detection of NE candidates which facilitates the manual validation. Indeed the validation requires expert intervention and it is faster and more reliable to proofread a rough annotation than to start from scratch. The second advantage of the automatic annotation is that it is a source of information for attribute selection.

We advise you to use the Alvis NLP line[1] for this step because it is reasonably fast, scalable, robust and reliable. Moreover D6.3NE tools uses the Alvis Enriched Document format to build file in fomats suitable for the next steps.

5 Manual validation

The objective is to build a corpus with a “perfect” annotation of NEs: a gold standard for machine learning. The more this step has been well prepared, the more accurate the results will be. This step eventually involves domain experts reading the corpus documents and annotating NEs, therefore it is the most costly step in terms of time and human resources.

The CADIXE[2] software was especially developed to annotate such texts; the D6.3NE provides scripts to translate Alvis Enriched Documents to a format suitable for CADIXE.

Ideally each document should be double-checked by two different experts and conflictual annotations arbitrated by the college of experts. It may seem unreasonable due to time and resources constraints, however a corpus with a clean annotation is crucial for machine learning. If you use a team of experts, we advise to write down a set of clear guidelines for them in order to obtain a homogeneous training set.

The D6.3NE package provides a directory framework to assist the organization of a double validation.

6 Machine Learning attributes and examples selection

Once the preceding step is finished, you have a gold standard at your disposal, which means you can feed the machine learning algorithms with clean training data.

Relevant attributes may be domain-specific, however we provide tools for preparing training data as generic as possible. There are five kinds of attributes handled by D6.3NE tools:

---

[1] url NLP

[2] http://caderige.imag.fr

1. a single document context attribute, named *title*;
2. a set of typographic attributes;
3. four dictionary attributes;
4. $2 \times N$ morphosyntactic context attributes, where N is the size of the window around the example;
5. a set of word context attributes, we distinguish the context before and after the example.

Table 1 outlines the meaning and the type of these attributes.

Attribute	**Type**	**Meaning**
Title	Boolean	Whether the example is in the title
Length	Numeric	Number of characters in the example
Other typographic	Boolean	Whether the example matches a given regular expression
Dictionary	Boolean	Whether the example is in a given dictionary
Dictionary inclusion	Boolean	Whether the example is a sub-string of a dictionary entry
POS before X	Modal	POS tag of the Xth word before the example
POS after X	Modal	POS tag of the Xth word before the example
Word X after	Boolean	Whether signal word X is present after the example
Word X before	Boolean	Whether signal word X is present before the example

Table 1: **Type and meaning of attributes**

The selection of most attributes is either straightforward or domain-specific but requires external resources. One needs a list of regular expressions for typographic attributes, which can be acquired by domain expertise. Optionally, you may build a dictionary and an ant-dictionary by using existing community resources. Finally signal word lists can be manually created through domain expertise, however you may miss a lot of relevant words. Another way is to systematically examine all words in the examples context and choose those who seem to discriminate the most NEs according to some kind of measure.

One tricky part is the selection of examples for machine learning. Positive examples are easy since the manual validation step provides certified NEs. In the other hand, there are main two strategies to select negative training examples. The first one is to pick near-misses. A near-miss is a word that is not NEs but looks like one or could be for some reason confused with one. The second strategy is to take all non-empty words.

D6.3NE provides tools to facilitate the signal word selection as well as the selection of relevant negative examples.

7 Learning

For machine learning purposes, we use the Weka software suite[3]. Weka implements a large span of machine learning techniques (cross-validation, IR eval-

---

[3] http://www.cs.waikato.ac.nz/m̃l/weka

uation...) and has a comprehensive library of algorithms (Bayesian, SVM, decision tree...).

The preferred training examples format file for Weka is called ARFF (Attribute-Relation File Format), the D6.3NE tools produce ARFF if you can provide all or part of the resources mentioned in the attribute selection step.

Part III

Data formats

This section describes the different data formats introduced with the D6.3NE. The paths to the files are relative to the root directory of the D6.3NE software package.

8 In-text Named Entity Annotation

The in-text named entity annotation file format was specifically designed to manually annotate named entities with the CADIXE software. A document in this format is an XML file valid to the `manual-ne-tag` DTD[4]. An Alvis Enriched Document can be translated in this format with the XSLT sheet `alvis2manual-ne-tag`[5]

The root element, called `document`, contains two elements. The first, `id`, indicates the document ID, it is the same as the `id` attribute of the `documentRecord` element in the Alvis Enriched Document. The second, `text`, contains the document text contents with NEs enclosed between `ne` tags, the rest of the text is left as is. The `ne` element accepts several attributes that correspond to Alvis Enriched Document linguistic features as shown in table 2. Additionally there's a `comment` attribute that allows annotators to record their remarks and doubts about each NE.

Attribute	**Alvis tag**
`canonical`	`canonical_form`
`type`	`named_entity_type`
`alvis-id`	`id`

Table 2: `manual-ne-tag` vs `enriched-document`

This format doesn't need to store explicitly the position of the NE in the text or the references to the included tokens since it is an *in-text* annotation format.

In order to render an XML file nicely, three files should be placed in CADIXE's directory tree: the document to edit in `manual-ne-tag` format, the corresponding DTD[6] and a rendering style sheet[7].

---

[4] `dtd/manual-ne-tag.dtd`

[5] `tools/alvis2manual-ne-tag.xslt`

[6] dtd/manual-ne-tag.dtd

[7] dtd/manual-ne-tag.css

9 In-Text Full Linguistic Annotation

The in-text full linguistic annotation format is a super-set of the in-text named entity annotation format. The main addition are `word` tags that enclose non-NE words. This tag accepts attributes so it can hold additional information such as POS-tag and lemma. The rest of the text, mainly spaces and punctuation, is left as is.

Documents in this format are mainly aimed at building machine learning examples. They are valid to the `full-la-tag` DTD[8]. You can generate them from an Alvis Enriched Document with the `alvis2full-la-tag` XSLT sheet[9]

10 Alvis Named Entities Differences

The Alvis named entities differences format is an XML format that follows the `alvis-ne-diff` DTD[10]. It has been designed to register differences between two Alvis Enriched Documents in NE annotations. It is useful to retrieve differences in NE annotations of the same document before and after the manual validation or the differences between two manual annotations.

The root element, `diff`, accepts two attributes: `reference-id` and `modified-id`, the documents IDs of the reference and modified documents respectively. In principle they should be the same since it is only useful to compare different annotations of the same documents. Below `diff`, a list of `added` elements denotes NEs present in the modified document but lacking in the reference document, and `removed` elements denote NEs present in the reference document but lacking in the modified document. Both elements accept a `id` attribute corresponding to the NE IDs in the compared Alvis files. Obviously the `id` in an `added` tag references a NE in the modified document whereas, in a `removed` tag, it references a NE in the reference document. Additionally a `added` element can contain a list of `overlap` tags; each one denotes a NE in the reference document whose boundaries overlap with the new named entities. In this way the addition of new NEs can be distinguished from the correction of NE boundaries.

The `alvis-ne-diff` XSLT sheet allows you to generate a difference document from two Alvis Enriched Documents: one of them is a *reference* document and the other one is the *modified* document. Alternatively the `manual-ne-tag2alvis.pl` script can build a difference file.

---

[8] `dtd/full-la-tag.dtd`

[9] tools/alvis2full-la-tag.xslt

[10] dtd/alvis-ne-diff.dtd

Part IV

Tools

11 XSLT sheets

XSLT is a language designed to manipulate XML documents. We used this language whenever we had to translate a format to another, especially if both formats are XML. XSLT processors are widely available on all platforms, the D6.3NE framework assumes the use of `xsltproc` from the `libxml` libraries.

To use any XSLT processor, you must provide an XML source document and a XSLT sheet. The processor will yield one or several documents resulting from the application of the sheet to the source. Additionally, XSLT sheets can accept named parameters.

The present section describes each XSLT sheet provided in the D6.3NE software package; what they do and which parameters they need.

11.1 alvis2full-la-tag.xslt

The `alvis2full-la-tag` sheet translates an Alvis Enriched Document into an In-Text Full Linguistic Annotation document. The source document is the Alvis Enriched Document with linguistic annotation (presumedly from the NLP line). The single yielded document is a valid `full-la-tag` XML document.

This sheet works without parameters.

11.2 alvis2manual-ne-tag.xslt

The `manual-ne-tag` sheet translates an Alvis Enriched Document into an In-Text Manual NE Annotation document. The source document is the Alvis Enriched Document with linguistic annotation (presumed from the NLP line). The single yielded document is a valid `manual-ne-tag` XML document ready for use with CADIXE.

This sheet works without parameters.

11.3 alvis-la-count.xslt

The `alvis-la-count` sheet is used to count linguistic features in an Alvis Enriched Document in order to get an overview of a corpus. The source document is a an Alvis Enriched Document with linguistic annotation, it yields the document ID, the number of tokens, words, NEs, sentences and terms.

The table 3 shows the different accepted parameters.

Parameter	**Accepted values**	**Default value**	**Effect**
`id`	{`yes`, `no`}	`yes`	Display the document ID
`nes`	{`yes`, `no`}	`no`	Count named entities
`ne-type`	string	empty	Limit the NE counting to the specified type
`quiet`	{`yes`, `no`}	`no`	Don't display labels
`sentences`	{`yes`, `no`}	`no`	Count sentences
`terms`	{`yes`, `no`}	`no`	Count terms
`tokens`	{`yes`, `no`}	`no`	Count tokens
`words`	{`yes`, `no`}	`no`	Count words

Table 3: `alvis-la-count.xslt` **parameters**

11.4 alvis-ne-diff-count.xslt

The `alvis-ne-diff-count` sheet is used to count differences contained in a Alvis Named Entities Differences file, in order to get an overview of the impact of the validation on the corpus. The source document is a `alvis-ne-diff` file, it yields the number of removed and added NEs (with or without overlap).

The table 4 shows the different accepted parameters.

Parameter	**Accepted values**	**Default value**	**Effect**
`quiet`	{`yes`, `no`}	`no`	Don't display labels
`added`	{`yes`, `no`}	`yes`	Count added NEs without overlap
`modified`	{`yes`, `no`}	`yes`	Count added NEs with overlaps
`removed`	{`yes`, `no`}	`yes`	Count removed NEs

Table 4: `alvis-ne-diff.xslt` **parameters**

11.5 alvis-ne-diff.xslt

The `alvis-ne-diff` sheet computes the differences in NE annotation between two Alvis Enriched Document files with linguistic annotations. It is assumed that aside NEs, both files are identical. Indeed differences cannot be computed if both files do not share the same word and sentence segmentation. The source is the modified Alvis Enriched Document, it yields a `alvis-ne-diff` file.

The only mandatory parameter, `reference-file`, specifies the path to the reference file.

11.6 extract-comments.xslt

The `extract-comments` sheet lists the NEs for which the `comment` attribute is not empty. This is useful in the validation process to watch annotations that may need an arbitration. The source is a `manual-ne-tag` document, it yields the NEs and their associated comment.

This sheet works without parameters.

12 Scripts

All scripts are written in Perl.

12.1 manual-ne-tag2alvis.pl

The `manual-ne-tag2alvis.pl` script translates a `manual-ne-tag` back to an Alvis Enriched Document. This script needs a reference Alvis Enriched Document in order to fill the non-NE linguistic annotations and to retrieve the right tokens for each NE. It basically copies the reference file but replaces all NEs by those found in the `manual-ne-tag` file.

Usage:

```
manual-ne-tag2alvis.pl [--diff DIFF_FILE] [--prefix PREFIX]
                 REFERENCE_FILE MODIFIED_FILE
```

`REFERENCE_FILE` is the path to the reference Alvis Enriched Document file and `MODIFIED_FILE` is path to the manual NE annotation file. The script writes the new Alvis Enriched Document to the standard output. Since it knows about the reference file, it can also create a `alvis-ne-diff` file through the `--diff` option. The `DIFF_FILE` argument to this option is the path to the file it should write the differences in.

The `manual-ne-tag2alvis.pl` script keeps the old ID from the reference file when the NE has been unchanged. It creates a new unique ID for new NEs with the prefix `named_entity_man`, you can change the prefix with the `--prefix` option.

12.2 sum.pl

Surprisingly there is no standard Unix utility to sum numbers in a file. The `sum.pl` script sums numbers from standard input, it accepts one number per line and yields the addition of all numbers. Additionally if the input consists of lines with several numbers split by a separator, `sum.pl` will output a sum for each column.

Usage:

```
sum.pl [--separator REGEXP] [--cumulative]
```

The `--separator` option allows you to specify a regular expression to be used as a column separator. By default it is any non-numeric character.

If the `--cumulative` option has been set, `sum.pl` will output the sum computed for each input line. That may useful to build cumulative distributions.

12.3 tag2arff.pl

The `tag2arff.pl` script builds an ARFF file for Weka from one or several `full-la-tag` file(s).

Usage:

```
tag2arff.pl [OPTIONS] TAG_FILE [TAG_FILES...]
```

In principle, the `TAG_FILE` arguments are the paths to the source `full-la-tag` files. We'll see that when the `--diff` option is set, these arguments are a bit different.

12.3.1 Action options

`--header`
This option causes the script to output the header part of the ARFF file.

`--data`
This option causes the script to output the data part of the ARFF file.

`--count`
This option inhibits any other output and prints the number of positive and negative examples found.

By default, without any of these three options, `tag2arff.pl` will not produce any output. You can build the ARFF in two steps: the header the the data. However both `--header` and `--data` options can be set together in order to produce a complete ARFF file.

12.3.2 Attribute selection options

`--range INT`
This option sets the number of words to look around each example for POS and signal word attributes. The script will never look beyond sentence boundaries. By default `--range` is set to 5.

`--following FILE`
This option sets the path to a file containing the following signal words. It should be a text file containing one word in each line. The script searches in this list for words (in lemmatized form) *after* each example.

`--preceding FILE`
Same as above for preceding signal words.

`--dict FILE`
This option sets the path to a NE dictionary file for dictionary and dictionary inclusion attributes. If the `--dict` option is not set, `tag2arff.pl` will skip these attributes. The format is a dictionary entry for each line.

`--anti FILE`
Same as above for anti-dictionary attributes.

`--title`
If this option is set, `tag2arff.pl` will set the value of the title attribute to true for all examples found. Otherwise it sets to false by default.

`--vocabulary`
If this option is set, all attribute selection options will be ignored except for `--range`. In this case `tag2arff.pl` will produce an ARFF file with only three attributes for each example shown in table 5. It is useful for selecting signal words.

Attribute	`Type`	`Meaning`
`position`	integer	Distance between the example and the word
`direction`	{following, preceding}	Direction of the word from the example
`word`	string	Lemma of the word
`class`	{ne, not_ne}	Whether the example is NE or not

Table 5: **Vocabulary attributes**

12.3.3 Example selection options

`--type TYPE`
This option sets the type of NEs to build positive examples. By default `tag2arff.pl` takes all NEs regardless of their type.

`--negative REGEXP`
If this option is set, the script will consider that all non-NE words that match the specified regular expression (Perl syntax) are negative examples.

`--anti FILE`
If this option is set, the script will consider that all non-NE nouns and adjectives in the anti-dictionary are negative examples. Note that this option has an effect both on attribute and example selection.

`--sink FILE`
This option sets the path to a “sink” file that has the same format as dictionaries. The script will consider that all non-NE words immediately preceding sink entries will be negative examples. This option corresponds to the context near-miss selection.

`--diff`
This option tells `tag2arff.pl` that it must select negatives from `alvis-ne-diff` files. It considers that removed NEs to be negative examples. If this option is set, the usage is slightly different than usual: each `full-la-tag` file name *must* be followed by the corresponding `alvis-ne-diff` file name.

`--full`
If this option is set, all example selection options will be ignored except for `--type`. In this case `tag2arff.pl` will consider that all non-NE noun phrases are negative examples. The script uses an heuristic for discovering noun phrases: it takes all word sequences whose POS tags are noun or adjective.

Part V

Framework

The D6.3NE software package contains a framework directory in order to use the provided tools according to the work flow presented in part II. The directory structure is practically empty since you must provide the corpus and the resources needed by the tools (for instance dictionaries). Figure 1 details the work flow, the data formats used successively and the tools required to translate, the external resources you must provide and the places where files are stored.

13 The corpus sub-directory

The data corresponding to the processing of a corpus is in the `corpus` sub-directory. You will find sub-directories that will hold the same corpus in different stages of the work flow and in different formats:

- `start`: the original corpus;
- `automatic`: automatically annotated corpus;
- `manual1`: the corpus manually validated once;
- `manual2`: the corpus manually validated twice;
- `learn`: the corpus in machine learning format.

Each one of these directories share a common set of conventions.

Single document The corpus is handled in a single document per file basis for practical reasons. It allows you to make corpus statistics based on documents (NEs per document, for instance). Additionally the error recovery is better when documents are held in different files.

File formats The files are stored in a directory named after the file format. For instance the automatically annotated corpus is stored in both Alvis Enriched Document format and Manual Annotation Tags, thus you will find them in the `automatic/alvis-enriched-document` and `automatic/manual-ne-tag` directories respectively.

Makefile Each directory contains a `Makefile` that allows you to fill it automatically whenever it is possible simply by typing `make` eventually followed by a target name.

The rest of this part is dedicated to the description of each directory/step.

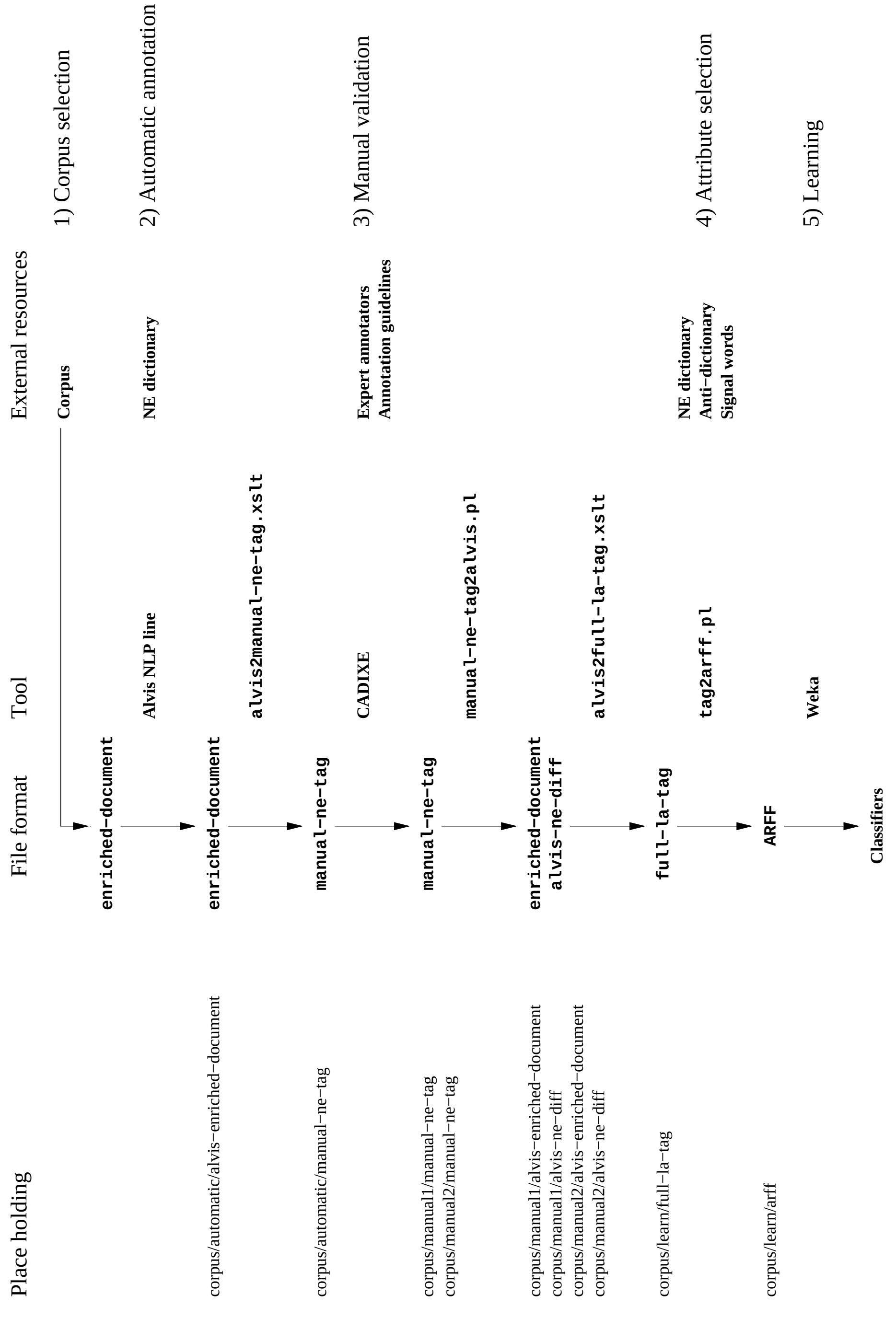

Figure 1: **Description of the framework**: the file format column shows the successive tranformations that will affect the corpus; the leftmost column indicates the path where the corpus will be held in each step; the *Tool* and *External resources* columns indicate the software and the resources used in each tranformation; the leftmost column anchors the transformations to a general workflow step.

13.1 `start`

This is where you should put your selected original corpus. The only necessary file format is the Alvis Enriched Document format. In order to have a maximum automation in the following steps, you should observe the following restrictions:

- each file contains a single `documentRecord` tag;
- the following tags must be filled: `documentRecord` (especially the `id` attribute), `canonicalDocument` and `originalDocument`, the other ones are optional though we encourage you to fill some meta-data;
- each file name ends with `.alvis`.

Exceptionally, there isn't any `Makefile` since you're the one who provides the corpus.

13.2 `automatic`

This directory holds the corpus automatically annotated with the Alvis NLP line. By typing `make` you launch the NLP line on each document and store the result in `alvis-enriched-document`. Additionally it translates each file in the `manual-ne-tag` format, ready for manual validation.

However in order to work properly you must have filled the `start` with your corpus and edited the `Makefile` to set the `NLPLINE_ROOT` variable. It should point to the path wherever you installed the NLP line. If you want to skip this step for any reason and directly make a manual annotation, leave `NLPLINE_ROOT` empty: it will copy the original corpus without annotating it.

13.3 `manual1`

The `manual1` directory will contain the corpus manually once validated by experts. You can send the automatically annotated corpus files to the experts in `manual-ne-tag` format, they validate thanks to the CADIXE software and send you back the corrected files. These obviously should be kept in the `manual-ne-tag` sub-directory. We advise you to take extra care of these files since they represent a lot of man-hours of work.

By typing `make`, you'll see a list of accepted targets:

- `check` will check that your experts have sent you files valid against the `manual-ne-tag` DTD;
- both `alvis` and `diff` will translate the experts files into Alvis Enriched Document and `alvis-ne-diff`;
- `check-diff` will ensure that the difference files are valid.

Type `make` followed by the target name to achieve the desired result. There is no need to edit the `Makefile`, all is ready.

13.4 `manual2`

The `manual2` directory will contain the corpus twice validated by experts. The process is the same than above, except that you send experts files from `manual1` instead of `automatic`. If you can't afford to have a second annotation and will be satisfied with just one, you will just have to copy the files from `manual1`.

Another difference is that both `alvis-enriched-document` and `alvis-ne-diff` directories contain two sub-directories called `vs_automatic` and `vs_manual1`. Indeed these formats are generated with the `manual-ne-tag2alvis.pl` tool that need a reference Alvis Enriched Document file. These two directories are filled whether we take as reference the automatic annotation or the first manual validation. This is particularly useful because you get differences of NE annotation between the first and second validation pass as well as the resulting differences against the initial automatic annotation.

By typing `make`, you'll see a list of accepted targets:

- `check` will check that your experts have sent you files valid against the `manual-ne-tag` DTD;
- `alvis-vs_automatic`, `diff-vs_automatic` and `vs_automatic` will translate the experts files into Alvis Enriched Document and `alvis-ne-diff` by taking the automatic annotation as reference;
- `alvis-vs_manual1`, `diff-vs_manual1` and `vs_manual1` will translate the experts files into Alvis Enriched Document and `alvis-ne-diff` by taking the first validation as reference;
- `check-diff` will ensure that the difference files are valid.

Type `make` followed by the target name to achieve the desired result. There is no need to edit the `Makefile`, all is ready.

13.5 `learn`

This directory will hold the corpus in formats suitable for machine learning, their experiments and classifiers. The `Makefile` is quite complex since it handles the building of training examples, the selection of signal words, the launching of machine learning experiments and the production of classifiers. It is necessary you edit it to set a couple of variables to suit your corpus:

VOCABULARY This variable lists the paths to training set configuration files. These sets will be specifically used to build a vocabulary of words surrounding NEs in order to select signal words.

The configuration files are in fact command-line options for `tag2arff.pl`. The default value, `resources/no-attributes_all-examples.cl`, should handle most of the corpora. However you must edit this file and replace `XXX` with the desired values; basically you must choose the range and the NE type to consider.

If you wish to experiment signal word selection for different ranges and different NE types, make as many copies of this file, edit them with the according values and add the file names to the `VOCABULARY` variable. Note that they must have the `.cl` extension and should be placed in the `resources` directory in order to benefit from the `make` automation.

EXPERIMENTS This variable is similar to the previous one but concerns learning examples for experiments and classifiers. The default files are also quite standard but you must fill in the values in place of `XXX`. Unfortunately you won't be able to set the signal word lists before you proceed their selection.

WEKA_JAR Set this variable to the path to the `weka.jar` file that comes with the Weka package. It will be used for signal word selection, experiments and classifiers.

JAVA_MEM In some cases Java doesn't allocate enough memory depending on the ML algorithm and the size of the corpus. If the ML steps gives an "Out of memory" error message, uncomment this line.

Let's do something now: if you type `make`, you'll see a list of valid targets. The first one you should trigger launches a preliminary task: the translation of the corpus in `full-la-tag` format which is very practical to build training examples with context-dependent attributes. Type `make full-la-tag`, it will take the annotation from `manual2` and fill the `full-la-tag` directory.

The next steps consist in selecting signal words, making experiments and creating classifiers:

Signal words selection The next target, `vocabulary`, will produce several lists of words that discriminate the best NEs from non-NE words. This is automatically achieved in three steps:

1. creation of training sets in ARFF format in the `arff/vocabulary` directory;
2. vectorisation of these training sets, it is a standard ML technique that translates a string attribute into a vector of boolean attributes;
3. calculation of most discriminant words by using Information Gain.

The results are stored in `results/selected-vocabulary` and intermediary files in `arff/vocabulary` for reference. For each file mentioned in the `VOCABULARY` variable, there are three results: one for the following signal words, one the preceding ones and a last one for signal words regardless their direction.

The results displays a score but it is not possible to automatically build a list since a bit of expert knowledge is necessary to validate the signal words. Once you've build the signal word lists you can complete the `.cl` files for the experiments.

Experiments Making experiments consists on testing several ML algorithms on different attribute sets in order to detect the best possible combination for the production phase. In the framework this is done in two stages: the creation of the examples, then the application of ML algorithms implemented by Weka on the examples.

The examples are created by the `tag2arff.pl` tool. It is automatically launched by `make` with the `examples` target. The result is an ARFF file for each `.cl` file mentioned in the `EXPERIMENTS` variable, they are stored in the `arff/examples` directory.

Now you can experiment on this data in two different manners: either you use the Weka experimenter GUI to load and play with the data, either you can type `make experiments` to run the Weka CLI on a couple of prepared algorithms. With the first solution (Weka GUI), you have full control on all experiment parameters however it requires you to have some mastering of Weka. With the second solution you'll be limited to the pre-made parameters. Three algorithms have been parametrized: naive Bayes, J48 decision trees (C4.5) and SMO (SVM). The evaluation is done by randomly splitting the data in a training set with two thirds of the examples and a test set with the remaining examples; this process is repeated ten times.

Because SVMs are longer in several orders of magnitude to run, they won't be launched with the `experiments` target but with the `smo-experiments`. The experiment results are stored in the `results/experiments` directory in the form of CSV files containing the IR scores of the ten repetitions for each algorithm; you should be able to open them with most spreadsheet programs.

Classifiers When you found the best algorithm and the most optimal settings, you may wish to generate the actual classifier. Again you can use the Weka GUI to have full control on the classifier generation process. However if you are satisfied with one of the algorithms we selected, you can generate them automatically by typing `make classifiers`.

The results are stored in the `results/classifiers` directory. There are two files for each algorithm: a `.classifier` file and a `.model` file. The `.classifier` file contains the classifier in a human readable format alongside with some performance results (IR scores, confusion matrix, etc.). The `.model` binary file may be used with the Weka CLI to classify another corpus provided they are in ARFF format.

14 The corpus_transcript sub-directory

The `corpus_transcript` directory contains a copy of the framework with all steps filled with the corpus used for learning classifiers for the domain of DNA transcription in *Bacillus subtilis*. The original corpus is the result of a query to the MedLine database[11]. We selected 422 NE-rich abstracts for manual

---

[11] http://www.ncbi.nlm.nih.gov/entrez

annotation amongst 2397 yielded by MedLine. The detail for biology NE process is detailed in the D6.3 report.