InterSystems
Creative data technology

Language Development Tools

The language development tools can be used to evaluate language models. These Python scripts allow to generate iKnow output in a human-readable format. Such output can be compared with the output of earlier versions to evaluate the differences, or it can give insight in the linguistic processing from input to output.

The tools are available in the “language_development” folder. This document describes their functionality and usage.

genRaw.py

Summary	
What?	generate output that contains the input sentence, attributes, entities (Concepts, Relations, PathRelevant entities), Paths and attribute spans
Usage	`python genRAW.py <input_directory> <output_directory> <language_code>` E.g. `python genRAW.py C:\corpus\input\ C:\corpus\output\ en` Note: Make sure to end the input and output directory with a backslash.
Parameters	Formatting of output Fixed input and output directories and language code

With this script you can **generate output** that contains the input sentence, attributes, entities (Concepts, Relations, PathRelevant entities), Paths and attribute spans.

The tool can generate output in **two types of formatting**. “OldStyle=True” uses D, S, C, R, PR and P as abbreviations for document, sentence, Concept, Relation, PathRelevant and Path respectively. It has unprintable characters as delimiters. “OldStyle=False” uses the full terms and has quotes and double spaces as delimiters. OldStyle=True is the default.

Setting the preferred type of output can be done in genRAW.py at line 23.

Below is the comparison of the two types for “This is not a long text, but a sentence of twelve words.”

OldStyle=True (default)

```
D□ input.txtc:\tmp\en_input\input.txt
S□ <This> □is not□ a long text, □but□ a sentence □of□ twelve words.
<attr type="negation" literal="is not" token="is not">
<attr type="measurement" literal="twelve words." token="twelve" value="twelve">
PR□<this>
R□is not
C□2□long text
R□but
C□1□sentence
R□of
C□2□twelve words
P□this is not long text but sentence of twelve words
```

OldStyle=False

```
<D name="input.txt" file="c:\ tmp\en_input\input.txt">
<S PathRelevant= "This"  Relation= "is not"  NonRelevant= "a"  Concept= "long
text,"  Relation= "but"  NonRelevant= "a"  Concept= "sentence"  Relation= "of"
Concept= "twelve words."  >
<attr type="negation" literal="is not" marker="is not">
<attr type="measurement" literal="twelve words." marker="twelve" value="twelve">
<PathRelevant  "this">
<Relation  "is not">
<NonRelevant  "a">
<Concept  "long text">
<Relation  "but">
<NonRelevant  "a">
<Concept  "sentence">
<Relation  "of">
<Concept  "twelve words">
<P  "this" "is not" "long text" "but" "sentence" "of" "twelve words">
```

Instead of entering the input and output directory and the language code as arguments when running the script, you can hard-code them in the script as in_path_par (input directory), out_path_par (output directory) and language_par (language code). Use forward slashes in the paths to the directories and do not forget such a slash at the end of the paths, e.g. “C:/tmp/input/”.

The two-character ISO language codes for the available language models are cs, de, en, es, fr, ja, nl, pt, ru, sv and uk.

genTrace.py

Summary	
What?	Generate output with information on unexposed labels (and – in the future - intermediate processing steps).
Usage	`python genTrace.py <input_directory> <output_directory> <language_code>` E.g. `python genTrace.py C:\corpus\input\ C:\corpus\output\ en` Note: Make sure to end the input and output directory with a backslash.
Parameters	Many parameters to tune which information appears in the output, such as the result of normalization, the result of pre-processing, detected attributes, applied rules etc.

This script outputs, per sentence, the tokens found (one token per line), the input sentence, the ***lexreps*** found with the **labels** that are used for the linguistic rules processing, a list of **rules** that are applied during the linguistic processing, the resulting entities and some more details about the **linguistic processing steps**.

(Lexreps are 'lexical representations', i.e. elements that are recognized as entries from the lexreps.csv file or unrecognized tokens that automatically get the default label of the language model. The default label is a Concept label in all our models.)

Below is an overview of the types of information that genTrace can output. The ones that are in italics do not appear in the output by default - mostly because the information is covered by other types already - but they can be obtained by extending the script.

LexrepCreated	The list of tokens in an input sentence.
NormalizeToken	The result of normalization: punctuation is stripped off from the end of a token. The input is converted into lower case. In Japanese, double-width alphanumeric characters are converted to single-width, and single-width Katakana characters are converted to double-width.
AttributeDetected	Capitalization is detected before normalization. Tokens with capital letters automatically get the attribute label CapitalInitial, CapitalAll or CapitalMixed.
PreprocessToken	The result of rules from the language model file prepro.csv. The Japanese model does not include such a file.
SentenceFound	The input sentence before normalization and pre-processing.
LexrepIdentified	The result of token lookup in lexreps.csv and regex.csv. A *lexrep* can consist of more than one token.

RuleApplication	Affected lexreps, phase number, rule, rule number and possibly comments for each rule from rules.csv that fires during processing, in the order of application during the processing.
RuleApplicationResult	The affected lexreps of a rule with the labels they have after applying the rule. A RuleApplicationResult is connected to a RuleApplication.
JoinResult	Some rules merge two or more lexreps into one. JoinResult comes after RuleApplication and RuleApplicationResult and shows the merged lexrep.
RulesComplete	At the end of the rules processing, RulesComplete shows the resulting labels for each lexrep.
AmbiguityResolved	At the end of the rules processing, AmbiguityResolved shows the resulting type for each lexrep (Concept, Relation, PathRelevant, NonRelevant, BeginConcept, EndConcept, BeginEndConcept, BeginRelation, EndRelation or BeginEndRelation) and (again) the remaining labels.
ConceptFiltered	Result of applying filter.csv, applied on lexreps after the rules processing.
MergingConcept	Shows a Concept, BeginConcept or EndConcept that is to become part of a larger Concept. Larger Concepts have at least 2 elements and comply with the following pattern: [BeginConcept]? [Concept]* [EndConcept]? Examples: BeginConcept+Concept, Concept+Concept+Concept, BeginConcept+EndConcept.
MergedConcept	The result of MergingConcept.
MergingRelation	Shows a Relation, BeginRelation or EndRelation that is to become part of a larger Relation. Larger Relations have at least 2 elements and comply with the following pattern: [BeginRelation]? [Relation]* [EndRelation]? Examples: Relation+Relation+Relation+Relation, BeginRelation+Relation, Relation+Relation+EndRelation.
MergedRelation	The result of MergingRelation.
MergedRelationNonrelevant	NonRelevant elements between Relations don’t hinder the process of MergingRelation. They are ignored.
SentenceComplete	Shows the sentence at the end of the linguistic processing as a sequence of entities with a type (Concept, Relation, PathRelevant or NonRelevant) and a value (text).

EntityVector	Only applicable in Japanese. See the Glossary for a description.
MergedKatakana	Only applicable in Japanese. Consecutive katakana characters are at first treated as separate lexreps and will appear as such under LexrepIdentified. Before the rules processing, they get merged into one “word”. The result is shown under MergedKatakana.
LabelKatakana	Only applicable in Japanese. The result of MergedKatakana gets a label.
TraceTime	The time it has taken to process the input. By default, the time is expressed in milliseconds. The option to output microseconds is available in the script.

Instead of entering the input and output directory and the language code as parameters when running the script, you can hard-code them in the script as in_path_par (input directory), out_path_par (output directory) and language_par (language code). Use forward slashes in the paths to the directories and do not forget such a slash at the end of the paths, e.g. “C:/tmp/input/.

The two-character ISO language codes for the available language models are cs, de, en, es, fr, ja, nl, pt, ru, sv and uk.

ref_testing.py

Summary	
What?	generate new output for the provided reference materials and compare it with the reference output
Usage	`python ref_testing.py` No arguments are required:the script works on the data in the reference_materials directory.
Parameters	None

The purpose of this script is to show effects and side-effects of changes in language models. It takes the files under reference_materials\input and generates output in reference_materials\new_output. It compares the new output files with the ones in reference_materials\reference and writes a general summary and a more detailed comparison report for each pair of files in reference_materials\reports.

Ref_testing.py always processes files of all supported languages.

There are 3 types of input, each requiring different action in case of changed output:

- Core: The output for these files is considered to be correct. Future updates of the language models must generate exactly the same output, unless additional features have been added. An update that generates differences will not be accepted until after manual review and approval.
- Test: These files contain examples for rules and other elements of the language models. If new output differs from the reference, check the comparison reports to make sure that the differences are improvements.
- Udct_test: These files contain items that occur in a small sample user dictionary (see reference_materials\udct_test_dictionaries). The test script loads the user dictionary of the concerned language to process each of the files. If there are differences in the new output, make sure that the differences are improvements and that they are not caused by ignoring the dictionary.