Alex Dialogue Systems Framework Documentation

Release 0.1

2012-2013, UFAL-DSG, MFF, CUNI, CZ and other contributors

August 18, 2015

Contents

Alex Dialogue Systems Framework or simply Alex is a set of algorithms, classes, and tools to facilitate building spoken dialogue systems.

CHAPTER 1

Contents

1.1 Index of manually written in source code tree documentation

1.1.1 Building the language model for the Public Transport Info telephone service (Czech)

```
*WARNING* To build the language model, you will need a machine with a lot of memory (more than 16GB R
```

The data

To build the domain specific language model, we use the approach described in Approach to bootstraping the domain specific language models. So far, we have collected this data:

1. selected out-of-domain data - more than 2000 sentences
2. bootstrap text - 289 sentences
3. indomain data - more than 9000 sentences (out of which about 900 of the sentences are used as development data)

Building the models

The models are built using the `build.py` script.

It requires to set the following variables:

```
bootstrap_text                  = "bootstrap.txt"
classes                         = "../data/database_SRILM_classes.txt"
indomain_data_dir               = "indomain_data"
```

The variables description:

- bootstrap_text - the bootstrap.txt file contains handcrafted in-domain sentences.
- classes - the `../data/database_SRILM_classes.txt` file is created by the `database.py` script in the `alex/applications/PublicTransportInfoCS/data` directory.
- indomain_data_dir - should include links to directories containing `asr_transcribed.xml` files with transcribed audio data

The process of building/re-building the LM is:

```
cd ../data
./database.py dump
cd ../lm
./build.py
```

Distributions of the models

The `final.*` models are large. Therefore, they should be distributed online on-demand using the `online_update` function. Please do not forget to place the models generated by the `./build.py` script on the distribution servers.

Reuse of build.py

The `build.py` script can be easily generalised to a different language or different text data, e.g. the in-domain data.

1.1.2 Description of resource files for ASR

This directory contains acoustic models for different languages and recording conditions. It is assumed that only one acoustic model per language will be build.

However, one can build different acoustic models for different recording settings, e.g. one for VOIP and the other for desktop mic recordings.

Up to now, only VOIP acoustic models have been trained.

1.1.3 Description of resource files for VAD

Please note that to simplify deployment of SDSs, the the VAD is trained to be language independent. That means that VAD classifies silence (noise, etc.) vs. all sounds in any language.

At this moment, the `alex/resources/vad/` has only VAD models build using VOIP audio signal. The created models include:

- GMM models
- NN models

More information about the process of creating the VAD models is available in Building a voice activity detector (VAD).

Please note that the NN VAD is much better compared to GMM VAD. Also `alex/resources/vad/` stores the models, but they should not be checked in the repository anymore. Instead, they should be on the online_update server and downloaded from it when they are updated. More on online update is available in Online distribution of resource files such as ASR, SLU, NLG models.

1.1.4 Public Transport Info, Czech - telephone service

Running the system at UFAL with the full UFAL access

There are multiple configuration that can used to run the system. In general, it depends on what components you want go use and on what telephone extension you want to run the system.

Within UFAL, we run the system using the following commands:

- `vhub_live` - deployment of our live system on our toll-free phone number, with the default configuration

- `vhub_live_b1` - a system deployed to backup the system above
- `vhub_live_b2` - a system deployed to backup the system above
- `vhub_live_kaldi` - a version of our live system explicitly using Kaldi ASR

To test the system we use:

- `vhub_test` - default test version of our system deployed on our test extension, logging locally into `../call_logs`
- `vhub_test_google_only` - test version of our system on our test extension, using Google ASR, TTS, Directions, logging locally into `../call_logs`
- `vhub_test_google_kaldi` - test version of our system on our test extension, using Google TTS, Directions, and Kaldi ASR, logging locally into `../call_logs`
- `vhub_test_hdc_slu` - default test version of our system deployed on our test extension, using HDC SLU, logging locally into `../call_logs`
- `vhub_test_kaldi` - default test version of our system deployed on our test extension, using KALDI ASR, logging locally into `../call_logs`
- `vhub_test_kaldi_nfs` - default test version of our system deployed on our test extension, using KALDI ASR and logging to NFS

Running the system without the full UFAL access

Users outside UFAL can run the system using the following commands:

- `vhub_private_ext_google_only` - default version of our system deployed on private extension specified in `private_ext.cfg`, using Google ASR, TTS, Directions, and KALDI ASR, logging locally into `../call_logs`
- `vhub_private_ext_google_kaldi` - default version of our system deployed on private extension specified in `private_ext.cfg`, using Google TTS, Directions, and KALDI ASR, logging locally into `../call_logs`

If you want to test the system on your private extension, then modify the `private_ext.cfg` config. You must set your SIP domain including the port, user login, and password (You can obtain a free extension at http://www.sipgate.co.uk). Please make sure that you do not commit your login information into the repository.

```
config = {
        'VoipIO': {
                # default testing extesion
                'domain':   "*:5066",
                'user':     "*",
                'password': "*",
        },
}
```

Also, you will have to create a "private" directory where you can store your private configurations. As the private default configuration is not part of the Git repository, please make your own empty version of the private default configuration as follows.

```
mkdir alex/resources/private
echo "config = {}" > alex/resources/private/default.cfg
```

1.1.5 UFAL Dialogue act scheme

The purpose of this document is to describe the structure and function of dialogue acts used in spoken dialogue systems developed at UFAL, MFF, UK, Czech Republic.

Definition of dialogue acts

In a spoken dialogue system, the observations and the system actions are represented by dialogue acts. Dialogue acts represent basic intents (such as inform, request, etc.) and the semantic content in the input utterance (e.g. type=hotel, area=east). In some cases, the value can be omitted, for example, where the intention is to query the value of a slot e.g. `request(food)`.

In the UFAL Dialogue Act Scheme (UDAS), a dialogue act (DA) is composed of one or more dialogue act items (DAI). A dialogue act item is defined as a tuple composed of a dialogue act type, a slot name, and the slot value. Slot names and slot values are domain dependent, therefore they can be many. In the examples which follows, the names of the slots and their values are drown form a information seeking application about restaurants, bars and hotels. For example in a tourist information domain, the slots can include "food" or "pricerange" and the values can be such as "Italian", "Indian" or "cheap", "midpriced", or "expensive".

This can be described in more formal way as follows:

```
DA = (DAI)+
DAI = (DAT, SN, SV)
DAT = (ack, affirm, apology, bye, canthearyou, confirm,
    iconfirm, deny, hangup, hello, help, inform, negate,
    notunderstood, null, repeat, reqalts, reqmore, request,
    restart, select, thankyou)
```

where SN denotes a slot name and SV denotes a slot value.

The idea of dialogue comes from the information state update (ISU) approach of defining a dialogue state. In ISU, a dialogue act is understood as a set of deterministic operations on a dialogue state which which result in a new updated state. In the UFAL dialogue act scheme, the update is performed on the slot level.

The following explains each dialogue act type:

```
ack          - "Ok" - back channel
affirm       - simple "Yes"
apology      - apology for misunderstanding
bye          - end of a dialogue - simple "Goodbye"
confirm      - user tries to confirm some information
canthearyou - system or user does not hear the other party
deny         - user denies some information
hangup       - the user hangs up
hello        - start of a dialogue - simple "Hi"
help         - request for help
inform       - user provides some information or constraint
negate       - simple "No"
null         - silence, empty sentence, something that is not possible to
                 interpret, does nothing
```

`null` It can be also used when converting a dialogue act item confusion network into an N-best list to hold all the probability mass connected with all dialogue acts which were not added to the N-best list. In other words probability mass of pruned DA hypotheses.

```
notunderstood - informs that the last input was not understood
repeat       - request to repeat the last utterance
irepeat      - repeats the last utterance
```

```
reqalts     - ask for alternatives
reqmore     - ask for more details
request     - user requests some information
restart     - request a restart of the dialogue
select      - user or the system wants the other party to select between
                two values for one slot
thankyou    - simple "Thank you"
```

NOTE: Having this set of acts we cannot confirm that something is not equal to something, e.g. `confirm(x!=y)` → `confirm(pricerange != 'cheap')` → "Isn't it cheap?" If we used `confirm(pricerange = 'cheap')` then it means "Is it cheap?" In both cases, it is appropriate to react in the same way e.g. `inform(pricerange='cheap')` or `deny(pricerange = 'cheap')`.

NOTE: Please note that all slot values are always placed in quotes `"`.

Dialogue act examples

This section presents examples of dialogue acts:

```
ack()                           'ok give me that one'
                                'ok great'

affirm()                        'correct'
                                'erm yeah'

appology()                      'sorry'
                                'sorry I did not get that'

bye()                           'allright bye'
                                'allright then bye'

canthearyou()                   'hallo'
                                'are you still there'

confirm(addr='main square')     'erm is that near the central the main square'
                                'is it on main square'

iconfirm(addr='main square')    'Ack, on main square,'


iconfirm(near='cinema')         'You want something near cinema'

deny(name='youth hostel')       'not the youth hostel'

deny(near='cinema')             'ok it doesn't have to be near the cinema'

hello()                         'hello'
                                'hi'
                                'hiya please'

help()                          'can you help me'


inform(='main square')          'main square'

inform(addr='dontcare')         'i don't mind the address'
```

```
inform(food='chinese')          'chinese'
                                'chinese food'
                                'do you have chinese food'

negate()                        'erm erm no i didn't say anything'
                                'neither'
                                'no'

null()                          '' (empty sentence)
                                'abraka dabra' (something not interpretable)

repeat()                        'can you repeat'
                                'could you repeat that'
                                'could you repeat that please'

reqalts()                       'and anything else'
                                'are there any other options'
                                'are there any others'

reqmore()                       'can you give me more dtails'

request(food)                   'do you know what food it serves'
                                'what food does it serve'

request(music)                  'and what sort of music would it play'
                                'and what type of music do they play in these bars'

restart()                       'can we start again please'
                                'could we start again'

select(food="Chinese")&select(food="Italian)
                                'do you want Chinese or Italian food'

thankyou()                      'allright thank you then i'll have to look somewhere else'
                                'erm great thank you'
```

If the system wants to inform that no venue is matching provided constraints, e.g. "There is no Chinese restaurant in a cheap price range in the city centre" the system uses the `inform(name='none')` dialogue acts as in

Utterance: There is no Chinese restaurant in a cheap price range in the city centre"

Dialogue act: `inform(name='none')&inform(venue_type='restaurant')&inform(food_type='Chinese')&i`

There are examples of dialogue acts composed of several DAIs:

```
reqalts()&thankyou()            'no thank you somewhere else please'

request(price)&thankyou()       'thank you and how much does it cost'
                                'thank you could you tell me the cost'

affirm()&inform(area='south')&inform(music='jazz')&inform(type='bar')&request(name)
                                'yes i'd like to know the name of the bar in the south part of town t
                                'yes please can you give me the name of the bar in the south part of

confirm(area='central')&inform(name='cinema')
                                'is the cinema near the centre of town'


deny(music='pop')&inform(music='folk')
```

```
                                    'erm i don't want pop music i want folk folk music'


hello()&inform(area='east')&inform(drinks='cocktails')&inform(near='park')&inform(pricerange='dontca
                                    'hi i'd like a hotel in the east of town by the park the price doesn
```

An example dialogue form tourist information domain is in the following table:

Turn	Transcription	Dialogue act
System	Hello. How may I help you?	`hello()`
User	Hi, I am looking for a restaurant.	`inform(venue="restaurant")`
System	What type of food would you like?	`request(food)`
User	I want Italian.	`inform(food="Italian")`
System	Did you say Italian?	`confirm(food="Italian")`
User	Yes	`affirm()`

Semantic Decoding and Ambiguity

Very often there are many ways as to map (to interpret) a natural utterance into a dialogue act, , some times because of natural ambiguity of a sentence – sometimes because of the speech recognition errors. Therefore, a semantic parser will generate multiple hypotheses. In this case, each hypothesis will be assigned a probability meaning the likelihood of being correct and the dialogue manager will resolve this ambiguity in the context of the dialogue (e.g. other sentences).

For example, the utterance "I wan an Italian restaurant erm no Indian" can be interpreted as:

```
inform(venue="restaurant")&inform(food="Italian")&deny(food=Indian)
```

or:

```
inform(venue="restaurant")&inform(food="Indian")
```

In the first case, the utterance is interpreted that the user wants Italian restaurant and does not want Indian. However, in the second case, the user corrected what he just mistakenly said (that he wants Indian restaurant).

Please remember that semantic parsers should interpret an utterance only on the information present in the sentence. It is up to the dialogue manager to interpret it in the context of the whole dialogue:

```
inform(type=restaurant)&inform(food='Chinese')
'I want a Chinese restaurant'

inform(food='Chinese')
'I would like some Chinese food'
```

In the first case, the user explicitly says that he/she is looking for a restaurant. However, in the second case, the user said that he/she is looking for some venue serving Indian food which can be both a restaurant or only a take-away.

Building a statistical SLU parser for a new domain

From experience, it appears that the easiest approach to build a statistical parser for a new domain is to start with build a handcrafted (rule based) parser. There are several practical reasons for that:

1. a handcrafted parser can serve as a prototype module for a dialogue system when no data is available,
2. a handcrafted parser can serve as a baseline for testing data driven parsers,
3. a handcrafted parser in information seeking applications, if well implemented, achieves about 95% accuracy on transcribed speech, which is close to accuracy of what the human annotators achieve,

4. a handcrafted parser can be used to obtain automatic SLU annotation which can be later hand corrected by humans.

To build a data driven SLU, the following approach is recommended:

1. after some data is collected, e.g. a prototype of dialogue system using a handcrafted parser, the audio from the collected calls is manually transcribed and then parsed using the handcrafted parser,
2. the advantage of using automatic SLU annotations is that they are easy to obtain and reasonably accurate only several percent lower to what one can get from human annotators.
3. if better accuracy is needed then it is better to fix the automatic semantic annotation by humans,
4. then a data driven parser is trained using this annotation

Note that the main benefit of data driven SLU methods comes from the ability to robustly handle erroneous input. Therefore, the data driven SLU should be trained to map **the recognised speech** to the dialogue acts (e.g. obtained by the handcrafted parser on the transcribed speech and then corrected by human annotator).

Comments

The previous sections described the general set of dialogue acts in UFAL dialogue systems. However, exact set of dialogue acts depends on a specific application domain and is defined by the domain specific semantic parser.

The only requirement is that all the output of a parser must be accepted by the dialogue manager developed for the particular domain.

Apendix A: UFAL Dialogue acts

Act	Description
`ack()`	back channel – simple OK
`affirm()`	acknowledgement - simple "Yes"
`apology()`	apology for misunderstanding
`bye()`	end of a dialogue
`canthearyou()`	signalling problem with communication channel or that there is an unexpected silence
`confirm(x=y)`	confirm that x equals to y
`iconfirm(x=y)`	implicitly confirm that x equals to y
`deny(x=y)`	denies some information, equivalent to `inform(x != y)`
`hangup()`	end of call because someone hungup
`hello()`	start of a dialogue
`help()`	provide context sensitive help
`inform(x=y)`	inform x equals to y
`inform(name=none)`	inform that "there is no such entity that ... "
`negate()`	negation - simple "No"
`notuderstood()`	informs that the last input was not understood
`null()`	silence, empty sentence, something that is not possible to interpret, does nothing
`repeat()`	asks to repeat the last utterance
`irepeat()`	repeats the last uttered sentence by the system
`reqalts()`	request for alternative options
`reqmore()`	request for more details bout the current option
`request(x)`	request for information about x
`restart()`	restart the dialogue, forget all provided info
`select(x=y)&select(x=z)`	select between two values of the same slot
`silence()`	user or the system does not say anything and remain silent
`thankyou()`	simply thank you

1.1.6 RepeatAfterMe (RAM) for Czech - speech data collection

This application is useful for bootstraping of speech data. It asks the caller to repeat sentences which are randomly sampled from a set of preselected sentences.

- The Czech sentences (`sentences_es.txt`) are from Karel Capek novels Matka and RUR, and the Prague's Dependency Treebank.
- The Spanish sentences (`sentences_es.txt`) are taken from the Internet

If you want to run `ram_hub.py` on some specific phone number than specify the appropriate extension config:

```
$ ./ram_hub.py -c ram_hub_LANG.cfg  ../../resources/private/ext-PHONENUMBER.cfg
```

After collection desired number of calls, use `copy_wavs_for_transcription.py` to extract the wave files from the `call_logs` subdirectory for transcription. The files will be copied into into `RAM-WAVs` directory.

These calls must be transcribed by the Transcriber or some similar software.

1.1.7 Building a SLU for the PTIen domain

Available data

At this moment, we only have data which were automatically generated using our handcrafted SLU (HDC SLU) parser on the transcribed audio. In general, the quality of the automatic annotation is very good.

The data can be prepared using the `prapare_data.py` script. It assumes that there exist the `indomain_data` directory with links to directories containing `asr_transcribed.xml` files. Then it uses these files to extract transcriptions and generate automatic SLU annotations using the PTIENHDCSLU parser from the `hdc_slu.py` file.

The script generates the following files:

- `*.trn`: contains manual transcriptions
- `*.trn.hdc.sem`: contains automatic annotation from transcriptions using handcrafted SLU
- `*.asr`: contains ASR 1-best results
- `*.asr.hdc.sem`: contains automatic annotation from 1-best ASR using handcrafted SLU
- `*.nbl`: contains ASR N-best results
- `*.nbl.hdc.sem`: contains automatic annotation from n-best ASR using handcrafted SLU

The script accepts `--uniq` parameter for fast generation of unique HDC SLU annotations. This is useful when tuning the HDC SLU.

The script also accepts `--fast` parameter for fast approximate preparation of all data. It approximates the HDC SLU output from an N-best list using output obtained by parsing the 1-best ASR result.

Building the models

First, prepare the data. Link the directories with the in-domain data into the `indomain_data` directory. Then run the following command:

```
./prepare_data.py
```

Second, train and test the models.

```
./train.py && ./test.py && ./test_bootstrap.py
```

Third, look at the `*.score` files or compute the interesting scores by running:

```
./print_scores.sh
```

Future work

- The `prepare_data.py` will have to use ASR, NBLIST, and CONFNET data generated by the latest ASR system instead of the logged ASR results because the ASR can change over time.
- Condition the SLU DialogueActItem decoding on the previous system dialogue act.

Evaluation

Evaluation of ASR from the call logs files

The current ASR performance computed on from the call logs is as follows:

```
Please note that the scoring is implicitly ignoring all non-speech events.

Ref: all.trn
Tst: all.asr
|=============================================================================================|
|            | # Sentences  |  # Words  |   Corr   |   Sub    |   Del    |   Ins    |   Err    |
|-------------------------------------------------------------------------------------------|
| Sum/Avg    |     9111     |   24728   |  56.15   |  16.07   |  27.77   |   1.44   |  45.28   |
|=============================================================================================|
```

The results above were obtained using the Google ASR.

Evaluation of the minimum number of feature counts

Using 9111 training examples, we found that pruning should be set to

- min feature count = 3
- min classifier count = 4

to prevent overfitting.

Cheating experiment: train and test on all data

Due to sparsity issue, the evaluation on proper test and dev sets suffers from sampling errors. Therefore, here we presents results when all data are used as training data and the metrics are evaluated on the training data!!!

Using the `./print_scores.sh` one can get scores for assessing the quality of trained models. The results from experiments are stored in the `old.scores.*` files. Please look at the results marked as `DATA ALL ASR - *`.

If the automatic annotations were correct, we could conclude that the F-measure of the HDC SLU parser on 1-best is higher wne compared to F-measure on N-best%. This is confusing as it looks like that the decoding from n-best lists gives worse results when compared to decoding from 1-best ASR hypothesis.

Evaluation of TRN model on test data

The TRN model is trained on transcriptions and evaluated on transcriptions from test data. Please look at the results marked as `DATA TEST TRN - *`. One can see that the performance of the TRN model on TRN test data is **NOT** 100 % perfect. This is probably due to the mismatch between the train and test data sets. Once more training data will be available, we can expect better results.

Evaluation of ASR model on test data

The ASR model is trained on 1-best ASR output and evaluated on the 1-best ASR output from test data. Please look at the results marked as `DATA TEST ASR - *`. The **ASR model scores significantly better** on the ASR test data when compared to *the HDC SLU parser* when evaluated on the ASR data. The improvement is about 20 % in F-measure (absolute). This shows that SLU trained on the ASR data can be beneficial.

Evaluation of NBL model on test data

The NBL model is trained on N-best ASR output and evaluated on the N-best ASR from test data. Please look at the results marked as `DATA TEST NBL - *`. One can see that using nblists even from Google ASR can help; though

only a little (about 1 %). When more data will be available, more test and more feature engineering can be done. However, we are more interested in extracting features from lattices or confusion networks.

Now, we have to wait for a working decoder generating *good* lattices. The OpenJulius decoder is not a suitable as it crashes unexpectedly and therefore it cannot be used in a real system.

1.1.8 Handling non-speech events in Alex

The document describes handling non-speech events in Alex.

ASR

The ASR can generate either:

- a valid utterance
- the ‘‘‘‘ empty sentence word to denote that the input was silence
- the `_noise_` word to denote that the input was some noise or other sound which is not a regular word
- the `_laugh_` word to denote that the input was laugh
- the `_ehm_hmm_` word to denote that the input was ehm or ehm sounds
- the `_inhale_` word to denote that the input was inhale sound
- the `_other_` word to denote that the input was something else that was lost during speech processing approximations such as N-best list enumeration or when the ASR did not provided any result. This is because we do not know what the input was and it can be both something important or worth ignoring. As such, it deserves special treatment in the system.

SLU

The SLU can generate either:

- a ordinary dialogue act
- the `null()` act which should be ignored by the DM, and the system should respond with `silence()`
- the `silence()` act which denote that the user was silent, a probably reasonable system response is `silence()` as well
- the `other()` act which denote that the input was something else that was lost during processing

The SLU should map:

- ‘‘‘‘ to `silence()` - silence will be processed in the DM
- `_noise_`, `_laugh_`, `_ehm_hmm_`, and `_inhale_` to `null()` - noise can be ignored in general
- `_other_` to `other()` - other hypotheses will be handled by the DM, mostly by responding “I did not get that. Can you ... ?”

DM

The DM can generate either:

- a normal dialogue act
- the `silence()` dialogue act

The DM should map:

- `null()` to `silence()` - because the `null()` act denote that the input should be ignored; however there is a problem with this, read the note below for current workaround for this
- `silence()` to `silence()` or a normal dialogue act - the DM should be silent or to ask the user "Are still there?"
- `other()` to `notunderstood()` - to show the user that we did not understood the input and that the input should be rephrased instead of just being repeated.

PROBLEM As of now, both handcrafted and trained SLUs cannot correctly classify the `other()` dialogue act. It has a very low recall for this DA. Instead of the `other()` DA it returns the `null()` DA. Therefore, the `null()` act is processed in DMs as if it was the `other()` DA **for now**.

1.1.9 Public Transport Info, English - telephone service

Description

This application provides information about public transport connections in New York using English language. Just say origin and destination stops and the application will find and tell you about the available connections. You can also specify a departure or arrival time if necessary. It offers bus, tram and metro city connections, and bus and train inter-city connections.

The application is available at the telephone number 1-855-528-7350.

You can also:

- ask for help
- ask for a "restart" of the dialogue and start the conversation again
- end the call - for example, by saying "Good bye."
- ask for repetition of the last sentence
- confirm or reject questions
- ask about the departure or destination station, or confirm it
- ask for the number of transits
- ask for the departure or arrival time
- ask for an alternative connection
- ask for a repetition of the previous connection, the first connection, the second connection, etc.

In addition, the application provides also information about:

- weather forecast
- current time

1.1.10 Public Transport Info, Czech - telephone service

Description

This application provides information about public transport connections in Czech Republic using the Czech language. Just say (in Czech) your origin and destination stops and the application will find and tell you about the available con-

nections. You can also specify a departure or arrival time if necessary. It offers bus, tram and metro city connections, and bus and train inter-city connections.

The application is available at the toll-free telephone number +420 800 899 998.

You can also:

- ask for help
- ask for a "restart" of the dialogue and start the conversation again
- end the call - for example, by saying "Good bye."
- ask for repetition of the last sentence
- confirm or reject questions
- ask about the departure or destination station, or confirm it
- ask for the number of transits
- ask for the departure or arrival time
- ask for an alternative connection
- ask for a repetition of the previous connection, the first connection, the second connection, etc.

In addition, the application provides also information about:

- weather forecast
- the current time

Representation of semantics

Suggestion (MK): It would be better to treat the specification of hours and minutes separately. When they are put together, all ways the whole time expression can be said have to be enumerated in the CLDB manually.

1.1.11 Building of acoustic models using HTK

In this document, we describe building of acoustic models using the HTK toolkit using the provided scripts. These acoustic models can be used with the *OpenJulius* ASR decoder.

We build a different acoustic model for a each language and acoustic condition pair – `LANG_RCOND`. At this time, we provide two sets of scripts for building English and Czech acoustic models using the VOIP data.

In general, the scripts can be described for the language and acoustic condition `LANG_RCOND` as follows:

```
./env_LANG_RCOND.sh          - includes all necessary training parameters: e.g. the train and test da
                               training options including cross word or word internal triphones, lang
./train_LANG_RCOND.sh        - performs the training of acoustic models
./nohup_train_LANG_RCOND.sh  - calls the training script using nohup and redirecting the output into
```

The training process stores some configuration files, the intermediate files, and final models and evaluations in the `model_LANG_RCOND` directory:

```
model_LANG_RCOND/config - config contains the language or recording specific configuration files
model_LANG_RCOND/temp
model_LANG_RCOND/log
model_LANG_RCOND/train
model_LANG_RCOND/test
```

Training models for a new language

Scripts for Czech and English are already created. If you need models for a new language, you can start by copying all the original scripts and renaming them so as to reflect the new language in their name (substitute *_en* or *_cs* with your new language code). You can do this by issuing the following command (we assume `$OLDLANG` is set to either *en* or *cs* and `$NEWLANG` to your new language code):

```
bash htk $ find . -name "*_$OLDLANG*" |
           xargs -n1 bash -c "cp -rvn \$1 \${1/_$OLDLANG/_$NEWLANG}" bash
```

Having done this, references to the new files' names have to be updated, too:

```
bash htk $ find . -name "*_$NEWLANG*" -type f -execdir \
           sed --in-place s/_$OLDLANG/_$NEWLANG/g '{}' \;
```

Furthermore, you need to adjust language-specific resources to the new language in the following ways:

`htk/model_voip_$NEWLANG/monophones0` List all the phones to be recognised, and the special *sil* phone.

`htk/model_voip_$NEWLANG/monophones1` List all the phones to be recognised, and the special *sil* and *sp* phones.

`htk/model_voip_$NEWLANG/tree_ques.hed` Specify phonetic questions to be used for building the decision tree for phone clustering (see *[HTKBook]*, Section 10.5).

`htk/bin/PhoneticTranscriptionCS.pl` You can start from this script or use a custom one. The goal is to implement the orthography-to-phonetics mapping to obtain sequences of phones from transcriptions you have.

`htk/common/cmudict.0.7a` and `htk/common/cmudict.ext` This is an alternative approach to the previous point – instead of programming the orthography-to-phonetics mapping, you can list it explicitly in a pronouncing dictionary.

Depending on the way you want to implement the mapping, you want to set `$OLDLANG` to either *cs* or *en*.

To make the scripts work with your new files, you will have to update references to scripts you created. All scripts are stored in the `htk/bin`, `htk/common`, and `htk` directories as immediate children, so you can make the substitutions only in these files.

Credits and the licence

The scripts are based on the HTK Wall Street Journal Training Recipe written by Keith Vertanen (http://www.keithv.com/software/htk/). His code is released under the new BSD licence. The licence note is at http://www.keithv.com/software/htk/. As a result we can re-license the code under the APACHE 2.0 license.

The results

- total training data for voip_en is about 20 hours
- total training data for voip_cs is about 8 hours
- mixtures - there is 16 mixtures is slightly better than 8 mixtures for voip_en
- there is no significant difference in alignment of transcriptions with -t 150 and -t 250
- the Julius ASR performance is about the same as of HDecode

- HDecode works well when cross word phones are trained, however the - performance of HVite decreases significantly
- when only word internal triphones are trained then the HDecode works, - however, its performance is worse than the HVite with a bigram LM
- word internal triphones work well with Julius ASR, do not forget disable CCD (it does not need context handling - though it still uses triphones)
- there is not much gain using the trigram LM in the Caminfo domain (about 1%)

1.1.12 Public Transport Info (Czech) – data

This directory contains the database used by the Czech Public Transport Info system, i.e. a list of public transportation stops, time expressions etc. that are understood by the system.

The main database module is located in `database.py`. You may obtain a dump of the database by running `./database.py dump`.

To build all needed generated files that are not versioned, run `build_data.sh`.

1.1.13 Contents of additional data files

Some of the data (for the less populous slots) is included directly in the code `database.py`, but most of the data (e.g., stops and cities) is located in additional list files.

Resources used by public transport direction finders

The sources of the data that are loaded by the application are:

- `cities.expanded.txt` – list of known cities and towns in the Czech Rep. (tab-separated: slot value name + possible surface forms separated by semicolons; lines starting with '#' are ignored)
- `stops.expanded.txt` – list of known stop names (same format)
- `cities_stops.tsv` – "compatibility table": lists compatible city-stops pairs, one entry per line (city and stop are separated by tabs). Only the primary stop and city names are used here.

The files `cities.expanded.txt` and `stops.expanded.txt` are generated from `cities.txt` and `stops.txt` using the `expand_stops.py` script (see documentation in the file itself; you need to have Morphodita Python bindings installed to successfully run this script). Please note that the surface forms in them are lowercased and do not include any punctuation (this can be obtained by setting the `-l` and `-p` parameters of the `expand_stops.py` script).

Colloquial stop names' variants that are added by hand are located in the `stops-add.txt` file and are appended to the `stops.txt` before performing the expansion.

Additional resources for the CRWS/IDOS directions finder

Since the CRWS/IDOS directions finder uses abbreviated stop names that need to be spelled out in ALEX, there is an additional resource file loaded by the system:

- `idos_map.tsv` – a mapping from the slot value names (city + stop) to abbreviated CRWS/IDOS names (stop list + stop)

The `convert_idos_stops.py` script is used to expand all possible abreviations and produce a mapping from/to the original CRWS/IDOS stop names as they appear, e.g., at the IDOS portal .

Resources used by the weather information service

The weather service uses one additional file:

- `cities_locations.tsv` – this file contains GPS locations of all cities in the Czech Republic.

1.1.14 Building a SLU for the PTIcs domain

Available data

At this moment, we only have data which were automatically generated using our handcrafted SLU (HDC SLU) parser on the transcribed audio. In general, the quality of the automatic annotation is very good.

The data can be prepared using the `prapare_data.py` script. It assumes that there exist the `indomain_data` directory with links to directories containing `asr_transcribed.xml` files. Then it uses these files to extract transcriptions and generate automatic SLU annotations using the PTICSHDCSLU parser from the `hdc_slu.py` file.

The script generates the following files:

- `*.trn`: contains manual transcriptions
- `*.trn.hdc.sem`: contains automatic annotation from transcriptions using handcrafted SLU
- `*.asr`: contains ASR 1-best results
- `*.asr.hdc.sem`: contains automatic annotation from 1-best ASR using handcrafted SLU
- `*.nbl`: contains ASR N-best results
- `*.nbl.hdc.sem`: contains automatic annotation from n-best ASR using handcrafted SLU

The script accepts `--uniq` parameter for fast generation of unique HDC SLU annotations. This is useful when tuning the HDC SLU.

Building the DAILogRegClassifier models

First, prepare the data. Link the directories with the in-domain data into the `indomain_data` directory. Then run the following command:

```
./prepare_data.py
```

Second, train and test the models.

:: cd ./dailogregclassifier

./train.py && ./test_trn.py && ./test_hdc.py && ./test_bootstrap_trn.py && ./test_bootsrap_hdc.py

Third, look at the `*.score` files or compute the interesting scores by running:

```
./print_scores.sh
```

Future work

- Exploit ASR Lattices instead of long NBLists.
- Condition the SLU DialogueActItem decoding on the previous system dialogue act.

Evaluation

Evaluation of ASR from the call logs files

The current ASR performance computed on from the call logs is as follows:

```
Please note that the scoring is implicitly ignoring all non-speech events.

Ref: all.trn
Tst: all.asr
|=============================================================================================|
|            | # Sentences  |  # Words  |   Corr   |   Sub    |   Del    |   Ins    |   Err    |
|-------------------------------------------------------------------------------------------|
| Sum/Avg    |     9111     |   24728   |  56.15   |  16.07   |  27.77   |   1.44   |  45.28   |
|=============================================================================================|
```

The results above were obtained using the Google ASR.

Evaluation of the minimum number of feature counts

Using 9111 training examples, we found that pruning should be set to

- min feature count = 3
- min classifier count = 4

to prevent overfitting.

Cheating experiment: train and test on all data

Due to sparsity issue, the evaluation on proper test and dev sets suffers from sampling errors. Therefore, here we presents results when all data are used as training data and the metrics are evaluated on the training data!!!

Using the `./print_scores.sh` one can get scores for assessing the quality of trained models. The results from experiments are stored in the `old.scores.*` files. Please look at the results marked as `DATA ALL ASR - *`.

If the automatic annotations were correct, we could conclude that the F-measure of the HDC SLU parser on 1-best is higher wne compared to F-measure on N-best%. This is confusing as it looks like that the decoding from n-best lists gives worse results when compared to decoding from 1-best ASR hypothesis.

Evaluation of TRN model on test data

The TRN model is trained on transcriptions and evaluated on transcriptions from test data. Please look at the results marked as `DATA TEST TRN - *`. One can see that the performance of the TRN model on TRN test data is **NOT** 100 % perfect. This is probably due to the mismatch between the train and test data sets. Once more training data will be available, we can expect better results.

Evaluation of ASR model on test data

The ASR model is trained on 1-best ASR output and evaluated on the 1-best ASR output from test data. Please look at the results marked as `DATA TEST ASR - *`. The **ASR model scores significantly better** on the ASR test data when compared to *the HDC SLU parser* when evaluated on the ASR data. The improvement is about 20 % in F-measure (absolute). This shows that SLU trained on the ASR data can be beneficial.

Evaluation of NBL model on test data

The NBL model is trained on N-best ASR output and evaluated on the N-best ASR from test data. Please look at the results marked as `DATA TEST NBL - *`. One can see that using nblists even from Google ASR can help; though only a little (about 1 %). When more data will be available, more test and more feature engineering can be done. However, we are more interested in extracting features from lattices or confusion networks.

Now, we have to wait for a working decoder generating *good* lattices. The OpenJulius decoder is not a suitable as it crashes unexpectedly and therefore it cannot be used in a real system.

1.1.15 Utils for building decoding graph HCLG

Summary

The `build_hclg.sh` script formats language model (LM) and acoustic model (AM) into files (e.g. HCLG) formated for Kaldi decoders.

The scripts extracts phone lists and sets from lexicon given the acoustic model (AM), the phonetic decision tree (tree) and the phonetic dictionary(lexicon).

The script silently supposes the same phone lists are generated from lexicon as the these used for training AM. If they are not the same, the script crashes.

The use case. Run the script with trained AM on full phonetic set for given language, pass the script also the tree used for tying the phonetic set and also give the script your LM and corresponding lexicon. The lexicon and the LM should also cover the full phonetic set for given language.

The `decode_indomain.py` script uses `HCLG.fst` and the rest of files generated by `build_hclg.sh` and performes decoding on prerecorded wav files. The reference speech transcription and path to the wav files are extracted from collected call logs. The wav files should be from one domain and the LM used to build `HCLG.fst` should be from the same domain. The `decode_indomain.py` also evaluates the decoded transcriptions. The Word Error Rate (WER), Real Time Factor (RTF) and other minor statistics are collected.

Dependencies of build_hclg.sh

The build_hclg.sh script requires the scripts listed belofw from `$KALDI_ROOT/egs/wsj/s5/utils`. The "utils scripts transitevely uses scripts from `$KALDI_ROOT/egs/wsj/s5/steps`. The dependency is solved in `path.sh` script which create corresponding symlinks and adds Kaldi binaries to your system path.

You just needed to set up `KALDI_ROOT` root variable and provide correct arguments. Try to run

Needed scripts from `utils` symlinked directory. * gen_topo.pl * add_lex_disambig.pl * apply_map.pl * eps2disambig.pl * find_arpa_oovs.pl * gen_topo.pl * make_lexicon_fst.pl * remove_oovs.pl * s2eps.pl * sym2int.pl * validate_dict_dir.pl * validate_lang.pl * parse_options.sh

Scripts from the list use Kaldi binaries, so you need Kaldi compiled on your system. The script `path.sh` adds Kaldi binaries to the `PATH` and also creates symlinks to `utils` and `steps` directories, where the helper scripts are located. You only need to set up `$KALDI_ROOT` variable.

1.1.16 Interective tests and unit tests

Testing of Alex can be divided into interactive tests, which depends on on some activity of a user e.g. calling a specific phone number or listening to some audi file, and unit tests, which are testing some very specific properties of algorithms or libraries.

Interactive tests

This directory contains only (interactive) tests, which can't be automated and the results must be verified by humans! E.g. playing or recording audio, testing VOIP connections.

Unit tests

Note that the unit tests should be placed in the same directory as the tested module and the name should be `test_*.py` e.g. `test_module_name.py`.

Using unittest module:

```
$ python -m unittest alex.test.test_string
```

This approach works everywhere but doesn't support test discovery.

Using nose test discovery framework, testing can largely automated. Nose searchs through packages and runs every test. Tests must be named `test_<something>.py` and must not be executable. Tests doesn't have to be run from project root, nose is able to find project root on its own.

How should my unit tests look like?

- Use unittest module
- Name the test file `test_<something>.py`
- Make the test file not executable

1.1.17 Approach to bootstraping the domain specific language models

```
**WARNING**: Please note that domain specific language models are build in ./alex/applications/*/lm
This text explains a simple approach to building a domain specific language models, which can be dif
domain.
```

While an acoustic model can be build domain independent, the language models (LMs) must be domain specific to ensure high accuracy of the ASR.

In general, building an in-domain LM is easy as long as one has enough of in-domain training data. However, when the in-domain data is scarce, e.g. when deploying a new dialogue system, this task is difficult and there is a need for some bootstrap solution.

The approach described here builds on:

1. some bootstrap text - probably handcrafted, which captures the main aspects of the domain
2. LM classes - which clusters words into classes, this can be derived from some domain ontology. For example, all food types belong to the FOOD class and all public transport stops stops belong to the STOP class
3. in-domain data - collected using some prototype or final system
4. general out-of-domain data - for example Wikipedia - from which is selected a subset of data, similar to our in-domain data

Then a simple process of building a domain specific language model can described as follows:

1. Append bootstrap text to the text extracted from the indomain data.
2. Build a class based language model using the data generated in the previous step and the classes derived from the domain ontology.
3. Score the general (domain independent) data using the LM build in the previous step.

4. Select some sentences with the lowest perplexity given the class based language model.
5. Append the selected sentences to the training data generated in the 1. step.
6. Re-build the class based language model.
7. Generate dictionaries.

Data for building general LMs

To get free general out-of-domain text data, we use the free W2C – Web to Corpus – Corpora available from the LINDAT project at: https://ufal-point.mff.cuni.cz/repository/xmlui/handle/11858/00-097C-0000-0022-6133-9

- English: https://ufal-point.mff.cuni.cz/repository/xmlui/bitstream/handle/11858/00-097C-0000-0022-6133-9/eng.txt.gz
- Czech: https://ufal-point.mff.cuni.cz/repository/xmlui/bitstream/handle/11858/00-097C-0000-0022-6133-9/ces.txt.gz

Structure of each domain scripts

Each of the projects should contain:

1. build.py - builds the final LMs, and computes perplexity of final LMs

Necessary files for the LM

For each domain the LM package should contain:

1. ARPA trigram language model (`final.tg.arpa`)
2. ARPA bigram language model (`final.bg.arpa`)
3. HTK wordnet bigram language model (`final.bg.wdnet`)
4. List of all words in the language model (`final.vocab`)
5. Dictionary including all words in the language model using compatible phone set with the language specific acoustic model (`final.dict` - without pauses and `final.dict.sp_sil` with short and long pauses)

CamInfoRest

For more details please see `alex.applications.CamInfoRest.lm.README`.

PTIcs

For more details please see Building the language model for the Public Transport Info telephone service (Czech).

1.1.18 Online distribution of resource files such as ASR, SLU, NLG models

Large binary files are difficult to store in git. Therefore, files such as resource files for ASR, SLU or NLG are distributed online and on-demand.

To use this functionality you have to use the `online_update(file_name)` function from the `alex.utils.config` package. The functions checks the file name whether it exists locally and it is up-to-date. If it is missing or it is old, then a new version from the server is downloaded.

The function returns name if the downloaded file which equal to input file name. As a result it is transparent in a way, that this function can be used everywhere a file name must be entered.

The server is set to `https://vystadial.ms.mff.cuni.cz/download/`; however, it can be changed using the `set_online_update_server(server_name)` function from inside a config file, e.g. the (first) default config file.

1.1.19 Building of acoustic models using KALDI

In this document, we describe building of acoustic models using the KALDI toolkit and the provided scripts. These acoustic models can be used with the *Kaldi* decoders and especially with the Python wrapper of `LatgenFasterDecoder` which is integrated with Alex.

We build a different acoustic model for a each language and acoustic condition pair – `LANG_RCOND`. At this time, we provide two sets of scripts for building English and Czech acoustic models using the VOIP data.

In general, the scripts can be described for the language and acoustic condition `LANG_RCOND` as follows:

Summary

- Requires KALDI installation and Linux environment. (Tested on Ubuntu 10.04, 12.04 and 12.10.) Note: We recommend Kaldi fork Pykaldi, because you will need it also for integrated Kaldi decoder to Alex.
- Recipes deployed with the Kaldi toolkit are located at `$KALDI_ROOT/egs/name_of_recipe/s[1-5]/`. This recipe requires to set up `$KALDI_ROOT` variable so it can use Kaldi binaries and scripts from `$KALDI_ROOT/egs/wsj/s5/`.

Details

- The recommended settings are stored at `env_LANG_RCONG.sh` e.g `env_voip_en.sh`
- We recommend to adjust the settings in file *env_LANG_RCONG_CUSTOM.sh‘* e.g. `env_voip_en_CUSTOM.sh`. See below. Do not commit this file to the git repository!
- Our scripts prepare the data to the expected format to `$WORK` directory.
- Experiment files are stored to `$EXP` directory.
- The symbolic links to `$KALDI_ROOT/wsj/s5/utils` and `$KALDI_ROOT/wsj/s5/steps` are automatically created.
- The files `path.sh`, `cmd.sh` are necessary to `utils` and `steps` scripts. Do not relocate them!
- Language model (LM) is either built from the training data using SRILM or specified in `env_LANG_RCOND.sh`.

Example of `env_voip_en_CUSTOM.sh`

```
# uses every utterance for the recipe every_N=10 is nice for debugging
export EVERY_N=1
# path to built Kaldi library and scripts
export KALDI_ROOT=/net/projects/vystadial/lib/kronos/pykaldi/kaldi

export DATA_ROOT=/net/projects/vystadial/data/asr/cs/voip/
export LM_paths="build0 $DATA_ROOT/arpa_bigram"
export LM_names="build0 vystadialbigram"

export CUDA_VISIBLE_DEVICES=0  # only card 0 (Tesla on Kronos) will be used for DNN training
```

Running experiments

Before running the experiments, check that:

- you have the Kaldi toolkit compiled: - http://github.com/UFAL-DSG/pykaldi (Recommended Kaldi fork, tested, necessary for further Alex integration) - http://sourceforge.net/projects/kaldi/ (alternative, main Kaldi repository) - In order to compile Kaldi we suggest:

```
# build openfst
pushd kaldi/tools
make openfst_tgt
popd
```

```
# download ATLAS headers
pushd kaldi/tools
make atlas
popd
```

```
# generate Kaldi makefile ``kaldi.mk`` and compile Kaldi
pushd kaldi/src
./configure
make && make test
popd
```

- you have SRILM compiled. (This is needed for building a language model) unless you supply your own LM in the ARPA format.)

```
pushd kaldi/tools
# download the srilm.tgz archive from http://www.speech.sri.com/projects/srilm/download.html
./install_srilm.sh
pushd
```

- the `train_LANG_RCOND` script will see the Kaldi scripts and binaries. Check for example that `$KALDI_ROOT/egs/wsj/s5/utils/parse_options.sh` is valid path.
- in `cmd.sh`, you switched to run the training on a SGE[*] grid if required (disabled by default) and `njobs` is less than number of your CPU cores.

Start the recipe by running `bash train_LANG_RCOND.sh`.

Extracting the results and trained models

The main script, `bash train_LANG_RCOND.sh`, performs not only training of the acoustic models, but also decoding. The acoustic models are evaluated during running the scripts and evaluation reports are printed to the standard output.

The `local/results.py exp` command extracts the results from the `$EXP` directory. It is invoked at the end of the `train_LANG_RCOND.sh` script.

If you want to use the trained acoustic model outside the prepared script, you need to build the `HCLG` decoding graph yourself. (See http://kaldi.sourceforge.net/graph.html for general introduction to the FST framework in Kaldi.) The `HCLG.fst` decoding graph is created by `utils/mkgraph.sh`. See `run.sh` for details.

Credits and license

The scripts were based on Voxforge KALDI recipe http://vpanayotov.blogspot.cz/2012/07/voxforge-scripts-for-kaldi.html . The original scripts as well as theses scripts are licensed under APACHE 2.0 license.

1.1.20 Building a voice activity detector (VAD)

This text described how to build a voice activity detector (VAD) for Alex. This work builds multilingual VAD. That means that we do not have VADs for individual languages but rather only one. It appears that NN VAD has the capacity to distinguish between non-speech and speech in any language.

As of now, we use VAD based on neural networks (NNs) implemented in the Theano toolkit. The main advantage that the same code can efficiently run both CPUs and GPUs and Theano implements automatic derivations. Automatic derivations is very useful especially when gradient descend techniques, such as stochastic gradient descent, are used for model parameters optimisation.

Old GMM code is still present but it may not work and its performance would be significantly worse that of the current NN implementation.

Experiments and the notes for the NN VAD

- testing is performed on randomly sampled data points (20%) from the entire set
- L2 regularisation must be very small, in addition it does not help much
- instead of MFCC, we use mel-filter banks coefficients only. It looks like the performance is the same or even better
- as of 2014-09-19 the best compromise between the model complexity and the performance appears to be.
  - 30 previous frames
  - 15 next frames
  - 512 hidden units
  - 4 hidden layers
  - tanh hidden layer activation
  - 4x amplification of the central frame compared to outer frames
  - discriminative pre-training
  - given this setup we get about 95.3 % frame accuracy on about 27 million of all data

Data

```
data_vad_sil     # a directory with only silence, noise data and its mlf file
data_voip_cs     # a directory where CS data reside and its MLF (phoneme alignment)
data_voip_en     # a directory where EN data reside and its MLF (phoneme alignment)
model_voip       # a directory where all the resulting models are stored.
```

Scripts

```
upload_models.sh                    # uploads all available models in ``model_voip`` onto the Alex
train_voip_nn_theano_sds_mfcc.py    # this is the main trainign script, see its help for more detai
bulk_train_nn_theano_mbo_31M_sgd.sh # script with curently ``optimal`` setting for VAD
```

Comments

To save some time especially for multiple experiments on the same data, we store preprocessed speech parametrisation. The speech parametrisation is stored because it takes about 7 hours to produce. However, it takes only 1 minute to load from a disk file. The `model_voip` directory stores this speech parametrisation in `*.npc` files. There fore if new data is added, then these NPC files must be deleted. If there are no NPC files then they are automatically generated from the available WAV files.

The `data_voip_{cs,en}` alignment files (mlf files) can be trained using scripts `alex/alex/tools/htk` or `alex/alex/tools/kaldi`. See the `train_voip_{cs,en}.sh` scripts in one of the directories. Note that the Kaldi scripts first store alignment in `ctm` format and later converts it to `mlf` format.

1.1.21 Public Transport Info (English) – data

This directory contains the database used by the English Public Transport Info system, i.e. a list of public transportation stops, number expressions etc. that are understood by the system.

The main database module is located in `database.py`. You may obtain a dump of the database by running `./database.py dump`.

To build all needed generated files that are not versioned, run `build_data.sh`.

1.1.22 Contents of additional data files

Some of the data (for the less populous slots) is included directly in the code `database.py`, but most of the data (e.g., stops and cities) is located in additional list files.

Resources used by public transport direction finders and weather service

The sources of the data that are loaded by the application are:

- `cities.expanded.txt` – list of known cities and towns in the USA. (tab-separated: slot value name + possible forms separated by semicolons; lines starting with '#' are ignored)
- `states.expanded.txt` – list of us state names (same format).
- `stops.expanded.txt` – list of known stop names (same format) in NY.
- `stops.expanded.txt` – list of known stop names (same format) in NY.
- `streets.expanded.txt` – list of known street names (same format)
- `boroughs.expanded.txt` – list of known borough names (same format)
- `cities.locations.csv` – tab separated list of known cities and towns, their state and geo location (longitude|latitude).
- `stops.locations.csv` – tab separated list of stops, their cities and geo location (longitude|latitude).
- `stops.borough.locations.csv` – tab separated list of stops, their boroughs and geo location (longitude|latitude).
- `streets.types.locations.csv` – tab separated list of streets, their boroughs and type (Avenue, Street, Court etc.)

All of these files are generated from `states-in.csv`, `cities-in.csv`, `stops-in.csv`, `streets-in.csv` and `boroughs-in.csv` located at `./preprocessing/resources` using the `expand_states_script.py`, `expand_cities_script.py`, `expand_stops_script.py`,

`expand_streets_script.py` and `expand_boroughs_script.py` script respectively. Please note that all forms in `*.expanded.txt` files are lowercased and do not include any punctuation.

Colloquial name variants that are added by hand are located in the `./preprocessing/resources/*-add.txt` files for each slot and are appended to the expansion process.

`build_data.sh` script is combining all the expansion scripts mentioned earlier into one process.

1.1.23 Public Transport Info, English - telephone service

Running the system at UFAL with the full UFAL access

There are multiple configuration that can used to run the system. In general, it depends on what components you want go use and on what telephone extension you want to run the system.

Within UFAL, we run the system using the following commands:

- `vhub_mta1` - deployment of our live system on a 1-855-528-7350 phone number, with the default configuration
- `vhub_mta2` - a system deployed to backup the system above
- `vhub_mta3` - a system deployed to backup the system above
- `vhub_mta_btn` - a system deployed to backup the system above accessible via web page http://alex-ptien.com

To test the system we use:

- `vhub_devel` - default devel version of our system deployed on our test extension, logging locally into `../call_logs`

Running the system without the full UFAL access

Users outside UFAL can run the system using the following commands:

- `vhub_private_ext_google_only_hdc_slu` - default version of our system deployed on private extension specified in `private_ext.cfg`, using HDC_SLU, Google ASR, TTS, Directions, logging locally into `../call_logs`
- `vhub_private_ext_google_kaldi_hdc_slu` - default version of our system deployed on private extension specified in `private_ext.cfg`, using HDC_SLU, Google TTS, Directions, and KALDI ASR, logging locally into `../call_logs`

If you want to test the system on your private extension, then modify the `private_ext.cfg` config. You must set your SIP domain including the port, user login, and password. Please make sure that you do not commit your login information into the repository.

```
config = {
        'VoipIO': {
                # default testing extesion
                'domain':   "*:5066",
                'user':     "*",
                'password': "*",
        },
}
```

Also, you will have to create a "private" directory where you can store your private configurations. As the private default configuration is not part of the Git repository, please make your own empty version of the private default configuration as follows.

```
mkdir alex/resources/private
echo "config = {}" > alex/resources/private/default.cfg
```

CHAPTER 2

Alex modules

2.1 alex package

2.1.1 Subpackages

alex.applications package

Subpackages

alex.applications.PublicTransportInfoCS package

Subpackages

alex.applications.PublicTransportInfoCS.data package

Submodules

alex.applications.PublicTransportInfoCS.data.add_cities_to_stops module A script that creates a compatibility table from a list of stops in a certain city and its neighborhood and a list of towns and cities.

Usage:

./add_cities_to_stops.py [-d "Main city"] stops.txt cities.txt cities_stops.tsv

alex.applications.PublicTransportInfoCS.data.add_cities_to_stops.**add_cities_to_stops**(*cities*, *stops*, *main_ci*

alex.applications.PublicTransportInfoCS.data.add_cities_to_stops.**get_city_for_stop**(*cities*, *stop*, *main_city*)

alex.applications.PublicTransportInfoCS.data.add_cities_to_stops.**load_list**(*filename*, *suppress_comments=False*, *cols=1*)

alex.applications.PublicTransportInfoCS.data.add_cities_to_stops.**main**()

alex.applications.PublicTransportInfoCS.data.autopath module self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing "pypy" is in sys.path.

If you modify the master "autopath.py" version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

alex.applications.PublicTransportInfoCS.data.convert_idos_stops module Convert stops gathered from the IDOS portal into structures accepted by the PublicTransportInfoCS application.

Usage:

./convert_idos_stops.py cities.txt idos_stops.tsv stops.txt cites_stops.tsv idos_map.tsv

Input: cities.txt = list of all cities idos_stops.tsv = stops gathered from IDOS (format: "list_id<t>abbrev_stop")

List ID is the name of the city for city public transit, "vlak" for trains and "bus" for buses.

Output: stops.txt = list of all stops (unabbreviated) cities_stops.tsv = city-to-stop mapping idos_map.tsv = mapping from (city, stop) pairs into (list_id, abbrev_stop) used by IDOS

alex.applications.PublicTransportInfoCS.data.convert_idos_stops.**expand_abbrevs**(*stop_name*)
Apply all abreviation expansions to the given stop name, all resulting variant names, starting with the 'main' variant.

alex.applications.PublicTransportInfoCS.data.convert_idos_stops.**expand_numbers**(*stop_name*)
Spell out all numbers that appear as separate tokens in the word (separated by spaces).

alex.applications.PublicTransportInfoCS.data.convert_idos_stops.**main**()

alex.applications.PublicTransportInfoCS.data.convert_idos_stops.**unambig_variants**(*variants*, *idos_list*)
Create 'unambiguos' variants for a stop name that equals a city name, depending on the type of the stop (bus, train or city public transit).

alex.applications.PublicTransportInfoCS.data.convert_idos_stops.**unify_casing_and_punct**(*stop_*
Unify casing of a stop name (if a second stop of the same name is encountered, let it have the same casing as the first one).

alex.applications.PublicTransportInfoCS.data.database module

alex.applications.PublicTransportInfoCS.data.download_data module

alex.applications.PublicTransportInfoCS.data.expand_stops module

alex.applications.PublicTransportInfoCS.data.get_cities_location module A script that collects the locations of all the given cities using the Google Geocoding API.

Usage:

./get_cities_locations.py [-d delay] [-l limit] [-a] cities_locations-in.tsv cities_locations-out.tsv

-d = delay between requests in seconds (will be extended by a random period up to 1/2 of the original value)

-l = limit maximum number of requests -a = retrieve all locations, even if they are set

alex.applications.PublicTransportInfoCS.data.get_cities_location.**get_google_coords**(*city*)
Retrieve (all possible) coordinates of a city using the Google Geocoding API.

alex.applications.PublicTransportInfoCS.data.get_cities_location.**random**() → x in the interval [0, 1).

alex.applications.PublicTransportInfoCS.data.ontology module

alex.applications.PublicTransportInfoCS.data.ontology.**add_slot_values_from_database**(*slot*, *category*, *exceptions=set*

alex.applications.PublicTransportInfoCS.data.ontology.**load_additional_information**(*fname*, *slot*, *keys*)

alex.applications.PublicTransportInfoCS.data.ontology.**load_compatible_values**(*fname*, *slot1*, *slot2*)

Module contents

alex.applications.PublicTransportInfoCS.hclg package

Submodules

alex.applications.PublicTransportInfoCS.hclg.autopath module self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing "pypy" is in sys.path.

If you modify the master "autopath.py" version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

alex.applications.PublicTransportInfoCS.hclg.kaldi_calibration module

Module contents

alex.applications.PublicTransportInfoCS.slu package

Subpackages

alex.applications.PublicTransportInfoCS.slu.dailogregclassifier package

Submodules

alex.applications.PublicTransportInfoCS.slu.dailogregclassifier.autopath module self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing "pypy" is in sys.path.

If you modify the master "autopath.py" version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

alex.applications.PublicTransportInfoCS.slu.dailogregclassifier.download_models module

alex.applications.PublicTransportInfoCS.slu.dailogregclassifier.test_bootstrap_trn module

alex.applications.PublicTransportInfoCS.slu.dailogregclassifier.test_trn module

alex.applications.PublicTransportInfoCS.slu.dailogregclassifier.train module

Module contents

alex.applications.PublicTransportInfoCS.slu.dainnclassifier package

Submodules

alex.applications.PublicTransportInfoCS.slu.dainnclassifier.autopath module self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing "pypy" is in sys.path.

If you modify the master "autopath.py" version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

alex.applications.PublicTransportInfoCS.slu.dainnclassifier.download_models module

alex.applications.PublicTransportInfoCS.slu.dainnclassifier.test_bootstrap_trn module

alex.applications.PublicTransportInfoCS.slu.dainnclassifier.test_trn module

Module contents

Submodules

alex.applications.PublicTransportInfoCS.slu.add_to_bootstrap module A simple script for adding new utterances along with their semantics to bootstrap.sem and bootstrap.trn.

Usage:

./add_to_bootsrap < input.tsv

The script expects input with tab-separated transcriptions + semantics (one utterance per line). It automatically generates the dummy 'bootstrap_XXXX.wav' identifiers and separates the transcription and semantics into two files.

`alex.applications.PublicTransportInfoCS.slu.add_to_bootstrap.`**`main`**`()`

alex.applications.PublicTransportInfoCS.slu.autopath module self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing "pypy" is in sys.path.

If you modify the master "autopath.py" version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

alex.applications.PublicTransportInfoCS.slu.consolidate_keyfiles module This scripts consolidates all input key files. That means, that it generates new keyfiles ({old_name}.pruned, which contains only entries common to all input ket files.

alex.applications.PublicTransportInfoCS.slu.consolidate_keyfiles.**main**()

alex.applications.PublicTransportInfoCS.slu.gen_bootstrap module

alex.applications.PublicTransportInfoCS.slu.gen_bootstrap.**confirm**(*f*, *v*, *c*)

alex.applications.PublicTransportInfoCS.slu.gen_bootstrap.**inform**(*f*, *v*, *c*)

alex.applications.PublicTransportInfoCS.slu.gen_bootstrap.**main**()

alex.applications.PublicTransportInfoCS.slu.gen_bootstrap.**zastavka**(*f*)

alex.applications.PublicTransportInfoCS.slu.gen_uniq module

alex.applications.PublicTransportInfoCS.slu.prepare_data module

alex.applications.PublicTransportInfoCS.slu.prepare_hdc_sem_from_trn module

alex.applications.PublicTransportInfoCS.slu.prepare_hdc_sem_from_trn.**hdc_slu**(*fn_input*, *constructor*, *fn_output*)

Use for transcription a HDC SLU model.

Parameters

- **fn_model** –
- **fn_input** –
- **constructor** –
- **fn_reference** –

Returns

alex.applications.PublicTransportInfoCS.slu.test_bootstrap_hdc module

alex.applications.PublicTransportInfoCS.slu.test_hdc module

alex.applications.PublicTransportInfoCS.slu.test_hdc.**hdc_slu_test**(*fn_input*, *constructor*, *fn_reference*)

Tests the HDC SLU.

Parameters

- **fn_model** –
- **fn_input** –
- **constructor** –
- **fn_reference** –

Returns

alex.applications.PublicTransportInfoCS.slu.test_hdc_utt_dict module

Module contents

Submodules

alex.applications.PublicTransportInfoCS.autopath module self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing "pypy" is in sys.path.

If you modify the master "autopath.py" version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

alex.applications.PublicTransportInfoCS.crws_enums module Various enums, semi-automatically adapted from the CHAPS CRWS enum list written in C#.

Comments come originally from the CRWS description and are in Czech.

alex.applications.PublicTransportInfoCS.crws_enums.**BEDS**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**CLIENTEXCEPTION_CODE**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**COMBFLAGS**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**COOR**
alias of Enum

class alex.applications.PublicTransportInfoCS.crws_enums.**CRCONST**

DELAY_CD = 'CD:'

DELAY_INTERN = 'X{0}_{1}:'

DELAY_INTERN_EXT = 'Y{0}_{1}:'

DELAY_TELMAX1 = 'TELMAX1:'

DELAY_ZSR = 'ZSR:'

EXCEPTIONEXCLUSION_CD = 'CD:'

alex.applications.PublicTransportInfoCS.crws_enums.**DELTAMAX**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**DEP_TABLE**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**EXFUNCTIONRESULT**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**FCS**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**LISTID**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**OBJECT_STATUS**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**REG**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**REMMASK**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**ROUTE_FLAGS**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**SEARCHMODE**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**ST**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**SVCSTATE**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**TIMETABLE_FLAGS**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**TRCAT**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**TRSUBCAT**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**TTDETAILS**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**TTERR**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**TTGP**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**TTINFODETAILS**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**TTLANG**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**VF**
alias of Enum

alex.applications.PublicTransportInfoCS.crws_enums.**enum**(***enums*)

alex.applications.PublicTransportInfoCS.cs_morpho module

alex.applications.PublicTransportInfoCS.directions module

alex.applications.PublicTransportInfoCS.exceptions module

exception alex.applications.PublicTransportInfoCS.exceptions.**PTICSHDCPolicyException**

Bases: *alex.components.dm.exceptions.DialoguePolicyException*

alex.applications.PublicTransportInfoCS.hdc_policy module

alex.applications.PublicTransportInfoCS.hdc_slu module

class alex.applications.PublicTransportInfoCS.hdc_slu.**DAIBuilder**(*utterance*, *abutterance_lenghts=None*)

Bases: object

Builds DialogueActItems with proper alignment to corresponding utterance words. When words are successfully matched using DAIBuilder, their indices in the utterance are added to alignment set of the DAI as a side-effect.

all_words_in(*words*)

any_phrase_in(*phrases*, *sub_utt=None*)

any_word_in(*words*)

build(*act_type=None*, *slot=None*, *value=None*)

Produce DialogueActItem based on arguments and alignment from this DAIBuilder state.

clear()

ending_phrases_in(*phrases*)

Returns True if the utterance ends with one of the phrases

Parameters **phrases** – a list of phrases to search for

Return type bool

first_phrase_span(*phrases*, *sub_utt=None*)

Returns the span (start, end+1) of the first phrase from the given list that is found in the utterance. Returns (-1, -1) if no phrase is found.

Parameters **phrases** – a list of phrases to be tried (in the given order)

Return type tuple

phrase_in(*phrase*, *sub_utt=None*)

phrase_pos(*words*, *sub_utt=None*)

Returns the position of the given phrase in the given utterance, or -1 if not found.

Return type int

class alex.applications.PublicTransportInfoCS.hdc_slu.**PTICSHDCSLU**(*preprocessing*, *cfg*)

Bases: *alex.components.slu.base.SLUInterface*

abstract_utterance(*utterance*)

Return a list of possible abstractions of the utterance.

Parameters **utterance** – an Utterance instance

Returns a list of abstracted utterance, form, value, category label tuples

handle_false_abstractions(*abutterance*)

Revert false positive alarms of abstraction

Parameters **abutterance** – the abstracted utterance

Returns the abstracted utterance without false positive abstractions

parse_1_best(*obs*, *verbose=False*, **args*, ***kwargs*)
Parse an utterance into a dialogue act.

:rtype DialogueActConfusionNetwork

parse_ampm(*abutterance*, *cn*)
Detects the ampm in the input abstract utterance.

Parameters

- **abutterance** – the input abstract utterance.
- **cn** – The output dialogue act item confusion network.

parse_city(*abutterance*, *cn*)
Detects stops in the input abstract utterance.

Parameters

- **abutterance** – the input abstract utterance.
- **cn** – The output dialogue act item confusion network.

parse_date_rel(*abutterance*, *cn*)
Detects the relative date in the input abstract utterance.

Parameters

- **abutterance** – the input abstract utterance.
- **cn** – The output dialogue act item confusion network.

parse_meta(*utterance*, *abutt_lenghts*, *cn*)
Detects all dialogue acts which do not generalise its slot values using CLDB.

NOTE: Use DAIBuilder ('dai' variable) to match words and build DialogueActItem, so that the DAI is aligned to corresponding words. If matched words are not supposed to be aligned, use PTICSHD-CSLU matching method instead. Make sure to list negative conditions first, so the following positive conditions are not added to alignment, when they shouldn't. E.g.: (not any_phrase_in(u, ['dobrý den', 'dobrý večer']) and dai.any_word_in("dobrý"))

Parameters

- **utterance** – the input utterance
- **cn** – The output dialogue act item confusion network.

Returns None

parse_non_speech_events(*utterance*, *cn*)
Processes non-speech events in the input utterance.

Parameters

- **utterance** – the input utterance
- **cn** – The output dialogue act item confusion network.

Returns None

parse_number(*abutterance*)
Detect a number in the input abstract utterance

Number words that form time expression are collapsed into a single TIME category word. Recognized time expressions (where FRAC, HOUR and MIN stands for fraction, hour and minute numbers respectively):

- •FRAC [na] HOUR
- •FRAC hodin*
- •HOUR a FRAC hodin*
- •HOUR hodin* a MIN minut*
- •HOUR hodin* MIN
- •HOUR hodin*
- •HOUR [0]MIN
- •MIN minut*

Words of NUMBER category are assumed to be in format parsable to int or float

Parameters **abutterance** (Utterance) – the input abstract utterance.

parse_stop(*abutterance*, *cn*)
Detects stops in the input abstract utterance.

Parameters

- **abutterance** – the input abstract utterance.
- **cn** – The output dialogue act item confusion network.

parse_task(*abutterance*, *cn*)
Detects the task in the input abstract utterance.

Parameters

- **abutterance** – the input abstract utterance.
- **cn** – The output dialogue act item confusion network.

parse_time(*abutterance*, *cn*)
Detects the time in the input abstract utterance.

Parameters

- **abutterance** – the input abstract utterance.
- **cn** – The output dialogue act item confusion network.

parse_train_name(*abutterance*, *cn*)
Detects the train name in the input abstract utterance.

Parameters

- **abutterance** –
- **cn** –

parse_vehicle(*abutterance*, *cn*)
Detects the vehicle (transport type) in the input abstract utterance.

Parameters

- **abutterance** – the input abstract utterance.
- **cn** – The output dialogue act item confusion network.

parse_waypoint(*abutterance*, *cn*, *wp_id*, *wp_slot_suffix*, *phr_wp_types*, *phr_in=None*)
Detects stops or cities in the input abstract utterance (called through parse_city or parse_stop).

Parameters

- **abutterance** – the input abstract utterance.
- **cn** – The output dialogue act item confusion network.
- **wp_id** – waypoint slot category label (e.g. "STOP=", "CITY=")
- **wp_slot_suffix** – waypoint slot suffix (e.g. "stop", "city")
- **phr_wp_types** – set of phrases for each waypoint type
- **phr_in** – phrases for 'in' waypoint type

alex.applications.PublicTransportInfoCS.hdc_slu.**all_words_in**(*utterance*, *words*)

alex.applications.PublicTransportInfoCS.hdc_slu.**any_phrase_in**(*utterance*, *phrases*)

alex.applications.PublicTransportInfoCS.hdc_slu.**any_word_in**(*utterance*, *words*)

alex.applications.PublicTransportInfoCS.hdc_slu.**ending_phrases_in**(*utterance*, *phrases*)
Returns True if the utterance ends with one of the phrases

Parameters

- **utterance** – The utterance to search in
- **phrases** – a list of phrases to search for

Return type bool

alex.applications.PublicTransportInfoCS.hdc_slu.**first_phrase_span**(*utterance*, *phrases*)
Returns the span (start, end+1) of the first phrase from the given list that is found in the utterance. Returns (-1, -1) if no phrase is found.

Parameters

- **utterance** – The utterance to search in
- **phrases** – a list of phrases to be tried (in the given order)

Return type tuple

alex.applications.PublicTransportInfoCS.hdc_slu.**phrase_in**(*utterance*, *words*)

alex.applications.PublicTransportInfoCS.hdc_slu.**phrase_pos**(*utterance*, *words*)
Returns the position of the given phrase in the given utterance, or -1 if not found.

Return type int

alex.applications.PublicTransportInfoCS.platform_info module

class alex.applications.PublicTransportInfoCS.platform_info.**CRWSPlatformInfo**(*crws_response*, *finder*)
Bases: object

find_platform_by_station(*to_obj*)

find_platform_by_train_name(*train_name*)

station_name_splitter = <_sre.SRE_Pattern object>

class alex.applications.PublicTransportInfoCS.platform_info.**PlatformFinderResult**(*platform*, *track*, *direction*)

Bases: object

class alex.applications.PublicTransportInfoCS.platform_info.**PlatformInfo**(*from_stop*, *to_stop*, *from_city*, *to_city*, *train_name*, *directions*)

Bases: object

alex.applications.PublicTransportInfoCS.platform_info_test module

class alex.applications.PublicTransportInfoCS.platform_info_test.**PlatformInfoTest**(*methodName='*

Bases: unittest.case.TestCase

test_matching()

alex.applications.PublicTransportInfoCS.preprocessing module

alex.applications.PublicTransportInfoCS.test_hdc_policy module

alex.applications.PublicTransportInfoCS.test_hdc_slu module

Module contents

alex.applications.PublicTransportInfoEN package

Subpackages

alex.applications.PublicTransportInfoEN.data package

Subpackages

alex.applications.PublicTransportInfoEN.data.preprocessing package

Submodules

alex.applications.PublicTransportInfoEN.data.preprocessing.compatibility_script_manual module A script that basically creates a csv file that contains a list of places from INPUT_FILE sith second column of a STRING_SAME_FOR_ALL and the benefit is that it can merge with already existing OUTPUT_FILE

unless -c flag is set.

Usage: /.compatibility_script_manual –name OUTPUT_FILE –main-place STRING_SAME_FOR_ALL –list INPUT_FILE [-c]

alex.applications.PublicTransportInfoEN.data.preprocessing.compatibility_script_manual.**hand**

alex.applications.PublicTransportInfoEN.data.preprocessing.compatibility_script_manual.**main**

alex.applications.PublicTransportInfoEN.data.preprocessing.compatibility_script_manual.**read**

alex.applications.PublicTransportInfoEN.data.preprocessing.compatibility_script_manual.**save**

alex.applications.PublicTransportInfoEN.data.preprocessing.compatibility_script_manual.**stic**

alex.applications.PublicTransportInfoEN.data.preprocessing.mta_to_csv module A script that takes mta stops file and it selects important fields and saves them (works with GTFS mainly) Usage:

./mta_to_csv.py [-m: main_city] [-o: output_file] stops.txt

alex.applications.PublicTransportInfoEN.data.preprocessing.mta_to_csv.**average_same_stops**(*sa*

alex.applications.PublicTransportInfoEN.data.preprocessing.mta_to_csv.**extract_fields**(*lines*, *header*, *main_ci* *skip_co*

alex.applications.PublicTransportInfoEN.data.preprocessing.mta_to_csv.**get_column_index**(*head* *cap-* *tion*, *de-* *fault*

alex.applications.PublicTransportInfoEN.data.preprocessing.mta_to_csv.**group_by_name**(*data*)

alex.applications.PublicTransportInfoEN.data.preprocessing.mta_to_csv.**load_list**(*filename*, *skip_comments*

alex.applications.PublicTransportInfoEN.data.preprocessing.mta_to_csv.**main**()

alex.applications.PublicTransportInfoEN.data.preprocessing.mta_to_csv.**remove_duplicities**(*li*

alex.applications.PublicTransportInfoEN.data.preprocessing.mta_to_csv.**remove_following_dupl**

alex.applications.PublicTransportInfoEN.data.preprocessing.mta_to_csv.**write_data**(*file_name*, *data*)

alex.applications.PublicTransportInfoEN.data.preprocessing.stops_to_streets_experiment module A script that takes mta stops, it splits them by special characters and each item takes for a street

alex.applications.PublicTransportInfoEN.data.preprocessing.stops_to_streets_experiment.**aver**

alex.applications.PublicTransportInfoEN.data.preprocessing.stops_to_streets_experiment.**extr**

alex.applications.PublicTransportInfoEN.data.preprocessing.stops_to_streets_experiment.**get_**

alex.applications.PublicTransportInfoEN.data.preprocessing.stops_to_streets_experiment.**grou**

alex.applications.PublicTransportInfoEN.data.preprocessing.stops_to_streets_experiment.**load**

alex.applications.PublicTransportInfoEN.data.preprocessing.stops_to_streets_experiment.**main**

alex.applications.PublicTransportInfoEN.data.preprocessing.stops_to_streets_experiment.**remo**

alex.applications.PublicTransportInfoEN.data.preprocessing.stops_to_streets_experiment.**remo**

alex.applications.PublicTransportInfoEN.data.preprocessing.stops_to_streets_experiment.**writ**

alex.applications.PublicTransportInfoEN.data.preprocessing.us_cities_to_csv module A script that takes us cities (city state_code)file and state-codes and it joins them

Usage:

./us_cities_to_csv.py [-o: output_file] cities.txt state-codes.txt

alex.applications.PublicTransportInfoEN.data.preprocessing.us_cities_to_csv.**average_same_c**

alex.applications.PublicTransportInfoEN.data.preprocessing.us_cities_to_csv.**extract_fields**

alex.applications.PublicTransportInfoEN.data.preprocessing.us_cities_to_csv.**get_column_inde**

alex.applications.PublicTransportInfoEN.data.preprocessing.us_cities_to_csv.**group_by_city_a**

alex.applications.PublicTransportInfoEN.data.preprocessing.us_cities_to_csv.**load_list**(*filenam*
skip_c

alex.applications.PublicTransportInfoEN.data.preprocessing.us_cities_to_csv.**load_state_code**

alex.applications.PublicTransportInfoEN.data.preprocessing.us_cities_to_csv.**main**()

alex.applications.PublicTransportInfoEN.data.preprocessing.us_cities_to_csv.**remove_duplicit**

alex.applications.PublicTransportInfoEN.data.preprocessing.us_cities_to_csv.**remove_followi**

alex.applications.PublicTransportInfoEN.data.preprocessing.us_cities_to_csv.**write_data**(*file_*
data)

Module contents

Submodules

alex.applications.PublicTransportInfoEN.data.autopath module self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing "pypy" is in sys.path.

If you modify the master "autopath.py" version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

alex.applications.PublicTransportInfoEN.data.database module

alex.applications.PublicTransportInfoEN.data.download_data module

alex.applications.PublicTransportInfoEN.data.expand_boroughs_script module A script that creates an expansion from a preprocessed list of boroughs

For usage write expand_boroughs_script.py -h

alex.applications.PublicTransportInfoEN.data.expand_boroughs_script.**all_to_lower**(*site_list*)

alex.applications.PublicTransportInfoEN.data.expand_boroughs_script.**handle_boroughs**(*boroughs_*
bor-
oughs_ou
bor-
oughs_ap
no_cache

alex.applications.PublicTransportInfoEN.data.expand_boroughs_script.**main**()

alex.applications.PublicTransportInfoEN.data.expand_cities_script module A script that creates an expansion from a preprocessed list of cities

For usage write expand_cities_script.py -h

alex.applications.PublicTransportInfoEN.data.expand_cities_script.**all_to_lower**(*site_list*)

alex.applications.PublicTransportInfoEN.data.expand_cities_script.**handle_cities**(*cities_in*,
cities_out,
cities_append,
no_cache=Fals

alex.applications.PublicTransportInfoEN.data.expand_cities_script.**main**()

alex.applications.PublicTransportInfoEN.data.expand_states_script module A script that creates an expansion from a preprocessed list of states

For usage write expand_states_script.py -h

alex.applications.PublicTransportInfoEN.data.expand_states_script.**handle_states**(*states_in*, *states_out*, *states_append*, *no_cache=Fals*

alex.applications.PublicTransportInfoEN.data.expand_states_script.**main**()

alex.applications.PublicTransportInfoEN.data.expand_stops_script module A script that creates an expansion from a list of stops

For usage write expand_stops_script.py -h

alex.applications.PublicTransportInfoEN.data.expand_stops_script.**append**(*major*, *minor*)

alex.applications.PublicTransportInfoEN.data.expand_stops_script.**expand_place**(*stop_list*)

alex.applications.PublicTransportInfoEN.data.expand_stops_script.**file_check**(*filename*, *message=u'reading file'*)

alex.applications.PublicTransportInfoEN.data.expand_stops_script.**get_column_index**(*header*, *caption*, *default*)

alex.applications.PublicTransportInfoEN.data.expand_stops_script.**hack_stops**(*stops*)

alex.applications.PublicTransportInfoEN.data.expand_stops_script.**handle_compatibility**(*file_in*, *file_ou*, *no_ca*

alex.applications.PublicTransportInfoEN.data.expand_stops_script.**handle_csv**(*csv_in*, *csv_out*, *no_cache=False*)

alex.applications.PublicTransportInfoEN.data.expand_stops_script.**load_list**(*filename*, *skip_comments=True*)

alex.applications.PublicTransportInfoEN.data.expand_stops_script.**main**()

alex.applications.PublicTransportInfoEN.data.expand_stops_script.**merge**(*primary*, *secondary*, *surpress_warning=True*)

alex.applications.PublicTransportInfoEN.data.expand_stops_script.**preprocess_line**(*line*)

alex.applications.PublicTransportInfoEN.data.expand_stops_script.**process_places**(*places_in*, *place_out*, *places_add*, *no_cache=Fals*

alex.applications.PublicTransportInfoEN.data.expand_stops_script.**read_compatibility**(*filename*)

alex.applications.PublicTransportInfoEN.data.expand_stops_script.**read_expansions**(*stops_expande*

alex.applications.PublicTransportInfoEN.data.expand_stops_script.**read_exports**(*filename*)

alex.applications.PublicTransportInfoEN.data.expand_stops_script.**read_first_column**(*filename*, *sur-*
press_warn

alex.applications.PublicTransportInfoEN.data.expand_stops_script.**read_two_columns**(*filename*)

alex.applications.PublicTransportInfoEN.data.expand_stops_script.**save_list**(*output_file*, *out-*
put_list)

alex.applications.PublicTransportInfoEN.data.expand_stops_script.**save_out**(*output_file*, *out-*
put_dict, *sep-*
a-
ra-
tor=u';
')

alex.applications.PublicTransportInfoEN.data.expand_streets_script module A script that creates an expansion from a list of stops

For usage write expand_stops_script.py -h

alex.applications.PublicTransportInfoEN.data.expand_streets_script.**main**()

alex.applications.PublicTransportInfoEN.data.ontology module

alex.applications.PublicTransportInfoEN.data.ontology.**add_slot_values_from_database**(*slot*, *cat-*
e-
gory, *ex-*
cep-
tions=set

alex.applications.PublicTransportInfoEN.data.ontology.**load_compatible_values**(*fname*, *slot1*, *slot2*)

alex.applications.PublicTransportInfoEN.data.ontology.**load_geo_values**(*fname*, *slot1*, *slot2*, *sur-*
press_warning=True)

alex.applications.PublicTransportInfoEN.data.ontology.**load_street_type_values**(*fname*, *sur-*
press_warning=Fa

Module contents

alex.applications.PublicTransportInfoEN.slu package

Submodules

alex.applications.PublicTransportInfoEN.slu.add_to_bootstrap module A simple script for adding new utterances along with their semantics to bootstrap.sem and bootstrap.trn.

Usage:

./add_to_bootsrap < input.tsv

The script expects input with tab-separated transcriptions + semantics (one utterance per line). It automatically generates the dummy 'bootstrap_XXXX.wav' identifiers and separates the transcription and semantics into two files.

alex.applications.PublicTransportInfoEN.slu.add_to_bootstrap.**main**()

alex.applications.PublicTransportInfoEN.slu.autopath module self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing "pypy" is in sys.path.

If you modify the master "autopath.py" version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

alex.applications.PublicTransportInfoEN.slu.consolidate_keyfiles module

alex.applications.PublicTransportInfoEN.slu.consolidate_keyfiles.**main**()

alex.applications.PublicTransportInfoEN.slu.gen_bootstrap module

alex.applications.PublicTransportInfoEN.slu.gen_bootstrap.**confirm**(*f*, *v*, *c*)
alex.applications.PublicTransportInfoEN.slu.gen_bootstrap.**inform**(*f*, *v*, *c*)

alex.applications.PublicTransportInfoEN.slu.gen_bootstrap.**main**()

alex.applications.PublicTransportInfoEN.slu.gen_bootstrap.**zastavka**(*f*)

alex.applications.PublicTransportInfoEN.slu.prepare_data module

alex.applications.PublicTransportInfoEN.slu.prepare_data.**main**()
alex.applications.PublicTransportInfoEN.slu.prepare_data.**normalise_semi_words**(*txt*)

alex.applications.PublicTransportInfoEN.slu.query_google module

alex.applications.PublicTransportInfoEN.slu.query_google.**main**()

alex.applications.PublicTransportInfoEN.slu.test_bootstrap module

Module contents

Submodules

alex.applications.PublicTransportInfoEN.autopath module self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing "pypy" is in sys.path.

If you modify the master "autopath.py" version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

alex.applications.PublicTransportInfoEN.directions module

class alex.applications.PublicTransportInfoEN.directions.**Directions**(***kwargs*)
Bases: *alex.applications.PublicTransportInfoEN.directions.Travel*

Ancestor class for transit directions, consisting of several routes.

class alex.applications.PublicTransportInfoEN.directions.**DirectionsFinder**
Bases: object

Abstract ancestor for transit direction finders.

get_directions(*from_city*, *from_stop*, *to_city*, *to_stop*, *departure_time=None*, *arrival_time=None*, *parameters=None*)
Retrieve the transit directions from the given stop to the given stop at the given time.

Should be implemented in derived classes.

class alex.applications.PublicTransportInfoEN.directions.**GoogleDirections**(*input_json={}*, ***kwargs*)
Bases: *alex.applications.PublicTransportInfoEN.directions.Directions*

Traffic directions obtained from Google Maps API.

class alex.applications.PublicTransportInfoEN.directions.**GoogleDirectionsFinder**(*cfg*)
Bases: *alex.applications.PublicTransportInfoEN.directions.DirectionsFinder*, *alex.tools.apirequest.APIRequest*

Transit direction finder using the Google Maps query engine.

get_directions(**args*, ***kwds*)
Get Google maps transit directions between the given stops at the given time and date.

The time/date should be given as a datetime.datetime object. Setting the correct date is compulsory!

map_vehicle(*vehicle*)
maps PTIEN vehicle type to GOOGLE DIRECTIONS query vehicle

class alex.applications.PublicTransportInfoEN.directions.**GoogleRoute**(*input_json*)
Bases: *alex.applications.PublicTransportInfoEN.directions.Route*

class alex.applications.PublicTransportInfoEN.directions.**GoogleRouteLeg**(*input_json*)
Bases: *alex.applications.PublicTransportInfoEN.directions.RouteLeg*

class alex.applications.PublicTransportInfoEN.directions.**GoogleRouteLegStep**(*input_json*)
Bases: *alex.applications.PublicTransportInfoEN.directions.RouteStep*

VEHICLE_TYPE_MAPPING = {u'FUNICULAR': u'cable_car', u'COMMUTER_TRAIN': u'train', u'INTERCITY_BUS

class alex.applications.PublicTransportInfoEN.directions.**Route**
Bases: object

Ancestor class for one transit direction route.

class alex.applications.PublicTransportInfoEN.directions.**RouteLeg**
Bases: object

One traffic directions leg.

class alex.applications.PublicTransportInfoEN.directions.**RouteStep**(*travel_mode*)
Bases: object

One transit directions step – walking or using public transport. Data members: travel_mode – TRANSIT / WALKING

- **For TRANSIT steps:** departure_stop departure_time arrival_stop arrival_time headsign – direction of the transit line vehicle – type of the transit vehicle (tram, subway, bus) line_name – name or number of the transit line
- **For WALKING steps:** duration – estimated walking duration (seconds)

MODE_TRANSIT = u'TRANSIT'

MODE_WALKING = u'WALKING'

class alex.applications.PublicTransportInfoEN.directions.**Travel**(***kwargs*)
Bases: object

Holder for starting and ending point (and other parameters) of travel.

get_minimal_info()
Return minimal waypoints information in the form of a stringified inform() dialogue act.

alex.applications.PublicTransportInfoEN.exceptions module

exception alex.applications.PublicTransportInfoEN.exceptions.**PTIENHDCPolicyException**
Bases: *alex.components.dm.exceptions.DialoguePolicyException*

alex.applications.PublicTransportInfoEN.hdc_policy module

class alex.applications.PublicTransportInfoEN.hdc_policy.**PTIENHDCPolicy**(*cfg*, *ontology*)
Bases: *alex.components.dm.base.DialoguePolicy*

The handcrafted policy for the PTI-EN system.

DEFAULT_AMPM_TIMES = {u'night': u'00:00', u'evening': u'18:00', u'pm': u'15:00', u'am': u'10:00', u'morning': u'06:

DESTIN = u'FINAL_DEST'

ORIGIN = u'ORIGIN'

backoff_action(*ds*)
Generate a random backoff dialogue act in case we don't know what to do.

Parameters **ds** – The current dialogue state

Return type *DialogueAct*

check_city_state_conflict(*in_city*, *in_state*)
Check for conflicts in the given city and state. Return an apology() DA if the state and city is incompatible.

Parameters

- **in_city** – city slot value
- **in_state** – state slot value

Return type *DialogueAct*

Returns apology dialogue act in case of conflict, or None

check_directions_conflict(*wp*)
Check for conflicts in the given waypoints. Return an apology() DA if the origin and the destination are the same, or if a city is not compatible with the corresponding stop.

Parameters **wp** – wayponts of the user's connection query

Return type *DialogueAct*

Returns apology dialogue act in case of conflict, or None

confirm_info(*tobe_confirmed_slots*)
Return a DA containing confirming only one slot from the slot to be confirmed. Confirm the slot with the most probable value among all slots to be confirmed.

Parameters **tobe_confirmed_slots** – A dictionary with keys for all slots that should be confirmed, along with their values

Return type *DialogueAct*

filter_iconfirms(*da*)
Filter implicit confirms if the same information is uttered in an inform dialogue act item. Also filter implicit confirms for stop names equaling city names. Also check if the stop and city names are equal!

Parameters **da** – unfiltered dialogue act

Returns filtered dialogue act

fix_stop_street_slots(*changed_slots*)

gather_connection_info(*ds*, *accepted_slots*)
Return a DA requesting further information needed to search for traffic directions and a dictionary containing the known information. Infers city names based on stop names and vice versa.

If the request DA is empty, the search for directions may be commenced immediately.

Parameters **ds** – The current dialogue state

Return type DialogueAct, dict

gather_time_info(*ds*, *accepted_slots*)
Handles if in_city specified it handles properly filled in_state slot. If needed, a Request DA is formed for missing in_state slot.

Returns Reqest DA and in_state If the request DA is empty, the search for current_time may be commenced immediately.

Parameters **ds** – The current dialogue state,

gather_weather_info(*ds*, *accepted_slots*)
Handles in_city and in_state to be properly filled. If needed, a Request DA is formed for missing slots to be filled.

Returns Reqest DA and WeatherPoint - information about the place If the request DA is empty, the search for weather may be commenced immediately.

Parameters **ds** – The current dialogue state,

get_accepted_mpv(*ds*, *slot_name*, *accepted_slots*)
Return a slot's 'mpv()' (most probable value) if the slot is accepted, and return 'none' otherwise. Also, convert a mpv of '*' to 'none' since we don't know how to interpret it.

Parameters

- **ds** – Dialogue state
- **slot_name** – The name of the slot to query
- **accepted_slots** – The currently accepted slots of the dialogue state

Return type string

get_an_alternative(*ds*)
Return an alternative route, if there is one, or ask for origin stop if there has been no route searching so far.

Parameters **ds** – The current dialogue state

Return type *DialogueAct*

get_confirmed_info(*confirmed_slots*, *ds*, *accepted_slots*)
Return a DA containing information about all slots being confirmed by the user (confirm/deny).

Update the current dialogue state regarding the information provided.

WARNING This confirms only against values in the dialogue state, however, it should (also in some cases) confirm against the results obtained from database, e.g. departure_time slot.

Parameters

- **ds** – The current dialogue state
- **confirmed_slots** – A dictionary with keys for all slots being confirmed, along with their values

Return type *DialogueAct*

get_connection_res_da(*ds*, *ludait*, *slots_being_requested*, *slots_being_confirmed*, *accepted_slots*, *changed_slots*, *state_changed*)
Handle the public transport connection dialogue topic.

Parameters **ds** – The current dialogue state

Return type *DialogueAct*

get_current_time(*in_city*, *in_state*, *longitude*, *latitude*)

get_current_time_res_da(*ds*, *accepted_slots*, *state_changed*)
Generates a dialogue act informing about the current time. :rtype: DialogueAct

get_da(*dialogue_state*)

The main policy decisions are made here. For each action, some set of conditions must be met. These conditions depends on the action.

Parameters **dialogue_state** – the belief state provided by the tracker

Returns a dialogue act - the system action

get_default_stop_for_city(*city*)
Return a 'default' stop based on the city name (main bus/train station).

Parameters **city** – city name (unicode)

Return type unicode

get_directions(*ds*, *route_type=u'true'*, *check_conflict=False*)
Retrieve Google directions, save them to dialogue state and return corresponding DAs.

Responsible for the interpretation of AM/PM time expressions.

Parameters

- **ds** – The current dialogue state
- **route_type** – a label for the found route (to be passed on to `say_directions()`)
- **check_conflict** – If true, will check if the origin and destination stops are different and issue a warning DA if not.

Return type *DialogueAct*

get_help_res_da(*ds*, *accepted_slots*, *state_changed*)

get_iconfirm_info(*changed_slots*)
Return a DA containing all needed implicit confirms.

Implicitly confirm all slots provided but not yet confirmed.

This include also slots changed during the conversation.

Parameters **changed_slots** – A dictionary with keys for all slots that have not been implicitly confirmed, along with their values

Return type *DialogueAct*

get_limited_context_help(*dialogue_state*)

get_requested_alternative(*ds*, *slots_being_requested*, *accepted_slots*)
Return the requested route (or inform about not finding one).

Parameters **ds** – The current dialogue state

Return type *DialogueAct*

get_requested_info(*requested_slots*, *ds*, *accepted_slots*)
Return a DA containing information about all requested slots.

Parameters

- **ds** – The current dialogue state
- **requested_slots** – A dictionary with keys for all requested slots and the correct return values.

Return type *DialogueAct*

get_weather(*ds*, *ref_point=None*)
Retrieve weather information according to the current dialogue state. Infers state names based on city names and vice versa.

Parameters **ds** – The current dialogue state

Return type *DialogueAct*

get_weather_res_da(*ds*, *ludait*, *slots_being_requested*, *slots_being_confirmed*, *accepted_slots*, *changed_slots*, *state_changed*)
Handle the dialogue about weather.

Parameters

- **ds** – The current dialogue state
- **slots_being_requested** – The slots currently requested by the user

Return type *DialogueAct*

interpret_time(*time_abs*, *time_ampm*, *time_rel*, *date_rel*, *lta_time*)
Interpret time, given current dialogue state most probable values for relative and absolute time and date, plus the corresponding last-talked-about value.

Returns the inferred time value + flag indicating the inferred time type ('abs' or 'rel')

Return type tuple(datetime, string)

process_directions_for_output(*dialogue_state*, *route_type*)
Return DAs for the directions in the current dialogue state. If the directions are not valid (nothing found), delete their object from the dialogue state and return apology DAs.

Parameters

- **dialogue_state** – the current dialogue state
- **route_type** – the route type requested by the user ("last", "next" etc.)

Return type *DialogueAct*

req_arrival_time(*dialogue_state*)
Return a DA informing about the arrival time the destination stop of the last recommended connection.

req_arrival_time_rel(*dialogue_state*)
Return a DA informing about the relative arrival time the destination stop of the last recommended connection.

req_departure_time(*dialogue_state*)
Generates a dialogue act informing about the departure time from the origin stop of the last recommended connection.

:rtype : DialogueAct

req_departure_time_rel(*dialogue_state*)
Return a DA informing the user about the relative time until the last recommended connection departs.

req_distance(*dialogue_state*)
Return a DA informing the user about the distance and number of stops in the last recommended connection.

req_duration(*dialogue_state*)
Return a DA informing about journey time to the destination stop of the last recommended connection.

req_from_stop(*ds*)
Generates a dialogue act informing about the origin stop of the last recommended connection.

TODO: this gives too much of information. Maybe it would be worth to split this into more dialogue acts
and let user ask for all individual pieces of information. The good thing would be that it would lead to longer dialogues.

:rtype : DialogueAct

req_num_transfers(*dialogue_state*)
Return a DA informing the user about the number of transfers in the last recommended connection.

req_time_transfers(*dialogue_state*)
Return a DA informing the user about transfer places and time needed for the trasfer in the last recommended connection.

req_to_stop(*ds*)
Return a DA informing about the destination stop of the last recommended connection.

reset_on_change(*ds*, *changed_slots*)
Reset slots which depends on changed slots.

Parameters

- **ds** – dialogue state
- **changed_slots** – slots changed in the last turn

select_info(*tobe_selected_slots*)
Return a DA containing select act for two most probable values of only one slot from the slot to be used for select DAI.

Parameters **tobe_selected_slots** – A dictionary with keys for all slots which the two most probable values should be selected

Return type *DialogueAct*

alex.applications.PublicTransportInfoEN.hdc_policy.**randbool**(*n*)
Randomly return True in 1 out of n cases.

Parameters **n** – Inverted chance of returning True

Return type Boolean

alex.applications.PublicTransportInfoEN.hdc_slu module

class alex.applications.PublicTransportInfoEN.hdc_slu.**DAIBuilder**(*utterance*, *abutterance_lenghts=None*)
Bases: object

Builds DialogueActItems with proper alignment to corresponding utterance words. When words are successfully matched using DAIBuilder, their indices in the utterance are added to alignment set of the DAI as a side-effect.

all_words_in(*words*)

any_phrase_in(*phrases*, *sub_utt=None*)

any_word_in(*words*)

build(*act_type=None*, *slot=None*, *value=None*)
Produce DialogueActItem based on arguments and alignment from this DAIBuilder state.

clear()

ending_phrases_in(*phrases*)
Returns True if the utterance ends with one of the phrases

Parameters **phrases** – a list of phrases to search for

Return type bool

first_phrase_span(*phrases*, *sub_utt=None*)
Returns the span (start, end+1) of the first phrase from the given list that is found in the utterance. Returns (-1, -1) if no phrase is found.

Parameters **phrases** – a list of phrases to be tried (in the given order)

Return type tuple

phrase_in(*phrase*, *sub_utt=None*)

phrase_pos(*words*, *sub_utt=None*)
Returns the position of the given phrase in the given utterance, or -1 if not found.

Return type int

class alex.applications.PublicTransportInfoEN.hdc_slu.**PTIENHDCSLU**(*preprocessing*, *cfg*)

Bases: *alex.components.slu.base.SLUInterface*

abstract_utterance(*utterance*)

Return a list of possible abstractions of the utterance.

Parameters **utterance** – an Utterance instance

Returns a list of abstracted utterance, form, value, category label tuples

handle_false_abstractions(*abutterance*)

Revert false positive alarms of abstraction

Parameters **abutterance** – the abstracted utterance

Returns the abstracted utterance without false positive abstractions

parse_1_best(*obs*, *verbose=False*, **args*, ***kwargs*)

Parse an utterance into a dialogue act.

:rtype DialogueActConfusionNetwork

parse_ampm(*abutterance*, *cn*)

Detects the ampm in the input abstract utterance.

Parameters

- **abutterance** – the input abstract utterance.
- **cn** – The output dialogue act item confusion network.

parse_borough(*abutterance*, *cn*)

Detects stops in the input abstract utterance.

Parameters

- **abutterance** – the input abstract utterance.
- **cn** – The output dialogue act item confusion network.

parse_city(*abutterance*, *cn*)

Detects stops in the input abstract utterance.

Parameters

- **abutterance** – the input abstract utterance.
- **cn** – The output dialogue act item confusion network.

parse_date_rel(*abutterance*, *cn*)

Detects the relative date in the input abstract utterance.

Parameters

- **abutterance** – the input abstract utterance.
- **cn** – The output dialogue act item confusion network.

parse_meta(*utterance*, *abutt_lenghts*, *cn*)

Detects all dialogue acts which do not generalise its slot values using CLDB.

NOTE: Use DAIBuilder ('dai' variable) to match words and build DialogueActItem, so that the DAI is aligned to corresponding words. If matched words are not supposed to be aligned, use PTICSHDCSLU matching method instead. Make sure to list negative conditions first, so the following positive

conditions are not added to alignment, when they shouldn't. E.g.: (not any_phrase_in(u, ['dobrý den', 'dobrý večer']) and dai.any_word_in("dobrý"))

Parameters

- **utterance** – the input utterance
- **cn** – The output dialogue act item confusion network.

Returns None

parse_non_speech_events(*utterance*, *cn*)
Processes non-speech events in the input utterance.

Parameters

- **utterance** – the input utterance
- **cn** – The output dialogue act item confusion network.

Returns None

parse_number(*abutterance*)
Detect a number in the input abstract utterance

Number words that form time expression are collapsed into a single TIME category word. Recognized time expressions (where FRAC, HOUR and MIN stands for fraction, hour and minute numbers respectively):

•FRAC [na] HOUR

•FRAC hodin*

•HOUR a FRAC hodin*

•HOUR hodin* a MIN minut*

•HOUR hodin* MIN

•HOUR hodin*

•HOUR [0]MIN

•MIN minut*

Words of NUMBER category are assumed to be in format parsable to int or float

Parameters **abutterance** (Utterance) – the input abstract utterance.

parse_state(*abutterance*, *cn*)
Detects state in the input abstract utterance.

Parameters

- **abutterance** – the input abstract utterance.
- **cn** – The output dialogue act item confusion network.

parse_stop(*abutterance*, *cn*)
Detects stops in the input abstract utterance.

Parameters

- **abutterance** – the input abstract utterance.
- **cn** – The output dialogue act item confusion network.

parse_street(*abutterance*, *cn*)
Detects street in the input abstract utterance.

Parameters

- **abutterance** – the input abstract utterance.
- **cn** – The output dialogue act item confusion network.

parse_task(*abutterance*, *cn*)
Detects the task in the input abstract utterance.

Parameters

- **abutterance** –
- **cn** – The output dialogue act item confusion network.

parse_time(*abutterance*, *cn*)
Detects the time in the input abstract utterance.

Parameters

- **abutterance** – the input abstract utterance.
- **cn** – The output dialogue act item confusion network.

parse_vehicle(*abutterance*, *cn*)
Detects the vehicle (transport type) in the input abstract utterance.

Parameters

- **abutterance** – the input abstract utterance.
- **cn** – The output dialogue act item confusion network.

parse_waypoint(*abutterance*, *cn*, *wp_id*, *wp_slot_suffix*, *phr_wp_types*, *phr_in=None*)
Detects stops or cities in the input abstract utterance (called through parse_city or parse_stop).

Parameters

- **abutterance** – the input abstract utterance.
- **cn** – The output dialogue act item confusion network.
- **wp_id** – waypoint slot category label (e.g. "STOP=", "CITY=")
- **wp_slot_suffix** – waypoint slot suffix (e.g. "stop", "city")
- **phr_wp_types** – set of phrases for each waypoint type
- **phr_in** – phrases for 'in' waypoint type

alex.applications.PublicTransportInfoEN.hdc_slu.**all_words_in**(*utterance*, *words*)

alex.applications.PublicTransportInfoEN.hdc_slu.**any_phrase_in**(*utterance*, *phrases*)

alex.applications.PublicTransportInfoEN.hdc_slu.**any_word_in**(*utterance*, *words*)

alex.applications.PublicTransportInfoEN.hdc_slu.**ending_phrases_in**(*utterance*, *phrases*)

Returns True if the utterance ends with one of the phrases

Parameters

- **utterance** – The utterance to search in
- **phrases** – a list of phrases to search for

Return type bool

alex.applications.PublicTransportInfoEN.hdc_slu.**first_phrase_span**(*utterance*, *phrases*)

Returns the span (start, end+1) of the first phrase from the given list that is found in the utterance. Returns (-1, -1) if no phrase is found.

Parameters

- **utterance** – The utterance to search in
- **phrases** – a list of phrases to be tried (in the given order)

Return type tuple

alex.applications.PublicTransportInfoEN.hdc_slu.**last_phrase_pos**(*utterance*, *words*)

Returns the last position of a given phrase in the given utterance, or -1 if not found.

Return type int

alex.applications.PublicTransportInfoEN.hdc_slu.**last_phrase_span**(*utterance*, *phrases*)

Returns the span (start, end+1) of the last phrase from the given list that is found in the utterance. Returns (-1, -1) if no phrase is found.

Parameters

- **utterance** – The utterance to search in
- **phrases** – a list of phrases to be tried (in the given order)

Return type tuple

alex.applications.PublicTransportInfoEN.hdc_slu.**phrase_in**(*utterance*, *words*)

alex.applications.PublicTransportInfoEN.hdc_slu.**phrase_pos**(*utterance*, *words*)

Returns the position of the given phrase in the given utterance, or -1 if not found.

Return type int

alex.applications.PublicTransportInfoEN.preprocessing module

class alex.applications.PublicTransportInfoEN.preprocessing.**PTIENNLGPreprocessing**(*ontology*)

Bases: *alex.components.nlg.template.TemplateNLGPreprocessing*

Template NLG preprocessing routines for English public transport information.

This serves for spelling out relative and absolute time expressions,

preprocess(*template*, *svs_dict*)

Preprocess values to be filled into an NLG template. Spells out temperature and time expressions and translates some of the values to English.

Parameters **svs_dict** – Slot-value dictionary

Returns The same dictionary, with modified values

spell_temperature(*value*, *interval*)

Convert a temperature expression into words (assuming nominative).

Parameters

- **value** – Temperature value (whole number in degrees as string), e.g. '1' or '-10'.
- **interval** – Boolean indicating whether to treat this as a start of an interval, i.e. omit the degrees word.

Returns temperature expression as string

spell_time_absolute(*time*)
Convert a time expression into words.

Parameters **time** – The 12hr numerical time value in a string, e.g. '08:05:pm'

Returns time string with all numerals written out as words

spell_time_relative(*time*)
Convert a time expression into words.

Parameters **time** – Numerical time value in a string, e.g. '8:05'

Returns time string with all numerals written out as words 0:15 will generate '15 minutes' and not '0 hours and 15 minutes'.

class alex.applications.PublicTransportInfoEN.preprocessing.**PTIENSLUPreprocessing**(**args*, ***kwargs*)
Bases: *alex.components.slu.base.SLUPreprocessing*

Extends SLUPreprocessing for some transformations:

normalise_utterance(*utterance*)

alex.applications.PublicTransportInfoEN.site_preprocessing module

alex.applications.PublicTransportInfoEN.site_preprocessing.**expand**(*element*, *spell_numbers=True*)

alex.applications.PublicTransportInfoEN.site_preprocessing.**expand_stop**(*stop*, *spell_numbers=True*)

alex.applications.PublicTransportInfoEN.site_preprocessing.**fix_ordinal**(*word*)

alex.applications.PublicTransportInfoEN.site_preprocessing.**spell_if_number**(*word*, *use_coupling*, *ordinal=True*)

alex.applications.PublicTransportInfoEN.test_hdc_policy module

class alex.applications.PublicTransportInfoEN.test_hdc_policy.**TestPTIENHDCPolicy**(*methodName='ru*
Bases: unittest.case.TestCase

get_clean_ds()

get_config()

get_directions_json()

setUp()

set_ds_connection_info(*to_stop='none'*, *from_stop='none'*, *to_city='none'*, *from_city='none'*)

set_ds_directions()

set_ds_street_connection_info(*to_street='none'*, *to_street2='none'*, *to_borough='none'*, *from_street='none'*, *from_street2='none'*, *from_borough='none'*)

test_gather_connection_info_combined()

test_gather_connection_info_from_street_to_stop()

test_gather_connection_info_from_streets_to_stops()

test_gather_connection_info_from_streets_to_stops2()

test_gather_connection_info_infer_from_borough()

test_gather_connection_info_infer_from_city()

test_gather_connection_info_infer_from_city_iconfirm()

test_gather_connection_info_infer_from_to_city()

test_gather_connection_info_infer_to_city()

test_gather_connection_info_request_from_stop()

test_gather_connection_info_request_from_street()

test_gather_connection_info_request_to_borough()

test_gather_connection_info_request_to_city()

test_gather_connection_info_request_to_stop()

test_gather_connection_info_request_to_stop_from_empty()

test_gather_connection_info_request_to_street()

test_gather_connection_info_street_infer_from_to_borough()

test_gather_connection_info_street_infer_to_borough()

test_interpret_time_empty()

test_interpret_time_in_twenty_minutes()

test_interpret_time_morning()

test_interpret_time_string_now()

test_interpret_time_tomorrow()

test_interpret_time_tomorrow_at_eight_pm()

test_req_arrival_time_abs()

test_req_arrival_time_rel_in_five_minutes()

test_req_departure_time_abs()

test_req_departure_time_rel_in_five_minutes()

test_req_departure_time_rel_missed()

test_req_departure_time_rel_now()

alex.applications.PublicTransportInfoEN.test_hdc_slu module

class alex.applications.PublicTransportInfoEN.test_hdc_slu.**TestPTIENHDCSLU**(*methodName='runTest'*)

Bases: unittest.case.TestCase

classmethod get_cfg()

classmethod setUpClass()

test_hour_and_a_half()

test_parse_borough_from()

test_parse_borough_from_to()

test_parse_borough_int()

test_parse_borough_to()

test_parse_form_street_to_stop()

test_parse_from_borough_from_street()

test_parse_from_street_street_to_street()

test_parse_from_to_city()

test_parse_half_an_hour()

test_parse_in_a_minute()

test_parse_in_an_hour()

test_parse_in_two_hours()

test_parse_next_connection_time()

test_parse_quarter_to_eleven()

test_parse_street_at_streets()

test_parse_street_from_street_to_streets()

test_parse_street_from_streets()

test_parse_street_from_streets_to_streets()

test_parse_street_to_streets()

test_parse_three_twenty_five()

test_parse_to_borough_to_street()

test_parse_to_city_to_stop()

test_parse_to_city_to_stop2()

test_parse_two_and_a_half()

test_parse_two_hours()

test_parse_two_hours_and_a_half()

test_parse_two_hours_and_a_quarter()

test_seventeen()

test_seventeen_fourteen_o_clock()

test_seventeen_zero_five()

test_ten_o_clock()

test_ten_p_m()

alex.applications.PublicTransportInfoEN.time_zone module

class alex.applications.PublicTransportInfoEN.time_zone.**GoogleTimeFinder**(*cfg*)

Bases: *alex.tools.apirequest.APIRequest*

get_time(*place=None*, *lat=None*, *lon=None*)

Get time information at given place

obtain_geo_codes(*place=u'New York'*)

: :return: Returns tuple (longitude, latitude) for given place. Default value for place is New York

parse_time(*response*)

class alex.applications.PublicTransportInfoEN.time_zone.**Time**
Bases: object

Module contents

alex.applications.utils package

Submodules

alex.applications.utils.weather module

class alex.applications.utils.weather.**OpenWeatherMapWeather**(*input_json*, *condition_transl*, *date=None*, *daily=False*, *celsius=True*)
Bases: *alex.applications.utils.weather.Weather*

class alex.applications.utils.weather.**OpenWeatherMapWeatherFinder**(*cfg*)
Bases: *alex.applications.utils.weather.WeatherFinder*, *alex.tools.apirequest.APIRequest*

Weather service using OpenWeatherMap (http://openweathermap.org)

get_weather(**args*, ***kwds*)
Get OpenWeatherMap weather information or forecast for the given time.

The time/date should be given as a datetime.datetime object.

load(*file_name*)

class alex.applications.utils.weather.**Weather**
Bases: object

class alex.applications.utils.weather.**WeatherFinder**
Bases: object

Abstract ancestor for transit direction finders.

get_weather(*time=None*, *daily=False*, *place=None*)
Retrieve the weather for the given time, or for now (if time is None).

Should be implemented in derived classes.

class alex.applications.utils.weather.**WeatherPoint**(*in_city=None*, *in_state=None*)
Bases: object

Module contents

Submodules

alex.applications.ahub module

alex.applications.autopath module

self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing "pypy" is in sys.path.

If you modify the master "autopath.py" version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

alex.applications.exceptions module

exception alex.applications.exceptions.**HubException**
Bases: *alex.AlexException*

exception alex.applications.exceptions.**SemHubException**
Bases: *alex.applications.exceptions.HubException*

exception alex.applications.exceptions.**TextHubException**
Bases: *alex.applications.exceptions.HubException*

exception alex.applications.exceptions.**VoipHubException**
Bases: *alex.applications.exceptions.HubException*

alex.applications.shub module

class alex.applications.shub.**SemHub**(*cfg*)
Bases: *alex.components.hub.hub.Hub*

SemHub builds a text based testing environment for the dialogue manager components.

It reads dialogue acts from the standard input and passes it to the selected dialogue manager. The output is the form dialogue acts.

hub_type = u'SHub'

input_da_nblist()
Reads an N-best list of dialogue acts from the input.

:rtype : confusion network

output_da(*da*)
Prints the system dialogue act to the output.

parse_input_da(*l*)
Converts a text including a dialogue act and its probability into a dialogue act instance and float probability.

The input text must have the following form: [prob] the dialogue act

run()
Controls the dialogue manager.

alex.applications.thub module

alex.applications.vhub module

alex.applications.webhub module

Module contents

alex.components package

Subpackages

alex.components.asr package

Submodules

alex.components.asr.autopath module self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing "pypy" is in sys.path.

If you modify the master "autopath.py" version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

alex.components.asr.base module

alex.components.asr.common module

`alex.components.asr.common.`**`asr_factory`**(*cfg*, *asr_type=None*)
Returns instance of specified ASR decoder in asr_type.

The ASR decoders are imported on the fly, because they need external non Python libraries.

`alex.components.asr.common.`**`get_asr_type`**(*cfg*)
Reads the ASR type from the configuration.

alex.components.asr.exceptions module

exception `alex.components.asr.exceptions.`**`ASRException`**
Bases: *alex.AlexException*

exception `alex.components.asr.exceptions.`**`JuliusASRException`**
Bases: *alex.components.asr.exceptions.ASRException*

exception `alex.components.asr.exceptions.`**`JuliusASRTimeoutException`**
Bases: *alex.components.asr.exceptions.ASRException*

exception `alex.components.asr.exceptions.`**`KaldiASRException`**
Bases: *alex.components.asr.exceptions.ASRException*

exception alex.components.asr.exceptions.**KaldiSetupException**
Bases: *alex.components.asr.exceptions.KaldiASRException*

alex.components.asr.google module

alex.components.asr.pykaldi module

alex.components.asr.test_utterance module

class alex.components.asr.test_utterance.**TestUttCNFeats**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Basic test for utterance confnet features.

test_empty_features()

class alex.components.asr.test_utterance.**TestUtterance**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests correct working of the Utterance class.

setUp()

test_index()

test_ngram_iterator()

class alex.components.asr.test_utterance.**TestUtteranceConfusionNetwork**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Tests correct working of the UtteranceConfusionNetwork class.

Test using $ python -m unittest test_utterance

test_conversion_of_confnet_into_nblist()

test_idx_zero()

test_ngram_iterator()

test_replace()

test_repr_basic()

test_repr_w_long_links()

alex.components.asr.utterance module

class alex.components.asr.utterance.**ASRHypothesis**
Bases: *alex.ml.hypothesis.Hypothesis*

This is the base class for all forms of probabilistic ASR hypotheses representations.

class alex.components.asr.utterance.**AbstractedUtterance**(*surface*)
Bases: *alex.components.asr.utterance.Utterance*, *alex.ml.features.Abstracted*

classmethod from_utterance(*utterance*)
Constructs a new AbstractedUtterance from an existing Utterance.

iter_triples()

iter_typeval()

join_typeval(*type_*, *val*)

classmethod make_other(*type_*)

other_val = (u'[OTHER]',)

phrase2category_label (*phrase*, *catlab*)
Replaces the phrase given by ‘phrase’ by a new phrase, given by ‘catlab’. Assumes ‘catlab’ is an abstraction for ‘phrase’.

replace (*orig*, *replacement*)
Replaces the phrase given by ‘orig’ by a new phrase, given by ‘replacement’.

replace_typeval (*orig*, *replacement*)
Replaces the phrase given by ‘orig’ by a new phrase, given by ‘replacement’.

class alex.components.asr.utterance.**Utterance** (*surface*)
Bases: object

find (*phrase*)
Returns the word index of the start of first occurrence of ‘phrase’ within this utterance. If none is found, returns -1.

Arguments: phrase – a list of words constituting the phrase sought

index (*phrase*)
Returns the word index of the start of first occurrence of ‘phrase’ within this utterance. If none is found, ValueError is raised.

Arguments: phrase – a list of words constituting the phrase sought

insert (*idx*, *val*)

isempty ()

iter_ngrams (*n*, *with_boundaries=False*)

iter_with_boundaries ()
Iterates the sequence [SENTENCE_START, word1, ..., wordlast, SENTENCE_END].

lower ()
Lowercases words of this utterance.

BEWARE, this method is destructive, it lowercases self.

replace (*orig*, *replacement*, *return_startidx=False*)
Analogous to the ‘str.replace’ method. If the original phrase is not found in this utterance, this instance is returned. If it is found, only the first match is replaced.

Arguments: orig – the phrase to replace, as a sequence of words replacement – the replacement in the same form return_startidx – if set to True, the tuple (replaced, orig_pos)

is returned where ‘replaced’ is the new utterance and ‘orig_pos’ is the index of the word where ‘orig’ was found in the original utterance. If set to False (the default), only the resulting utterance is returned.

replace2 (*start*, *end*, *replacement*)
Replace the words from start to end with the replacement.

Parameters

- **start** – the start position of replaced word sequence
- **end** – the end position of replaced word sequence
- **replacement** – a replacement

Returns return a new Utterance instance with the word sequence replaced with the replacement

replace_all(*orig*, *replacement*)
Replace all occurrences of the given words with the replacement. Only replaces at word boundaries.

Parameters

- **orig** – the original string to be replaced (as string or list of words)
- **replacement** – the replacement (as string or list of words)

Return type *Utterance*

utterance

class alex.components.asr.utterance.**UtteranceConfusionNetwork**(*rep=None*)
Bases: *alex.components.asr.utterance.ASRHypothesis*, *alex.ml.features.Abstracted*

Word confusion network

Attributes:

cn: a list of alternatives of the following signature [word_index-> [alternative]]

XXX Are the alternatives always sorted wrt their probability in decreasing order?

TODO Define a lightweight class SimpleHypothesis as a tuple (probability, fact) with easy-to-read indexing. namedtuple might be the best choice.

class Index(*is_long_link*, *word_idx*, *alt_idx*, *link_widx*)
Bases: tuple

unique index into the confnet

Attributes: is_long_link – indexing to a long link? word_idx – first index either to self.cn or self._long_links alt_idx – second index ditto link_widx – if is_long_link, this indexes the word within a phrase of

the long link

alt_idx
Alias for field number 2

is_long_link
Alias for field number 0

link_widx
Alias for field number 3

word_idx
Alias for field number 1

class UtteranceConfusionNetwork.**LongLink**(*end*, *orig_probs*, *hyp*, *normalise=False*)
Bases: object

attrs = (u'end', u'orig_probs', u'hyp', u'normalise')
Represents a long link in a word confusion network.
Attributes: end – end index of the link (exclusive) orig_probs – list of probabilities associated with the ordinary
words this link corresponds to

hyp – a (probability, phrase) tuple, the label of this link. 'phrase' itself is a sequence of words (list of strings).
normalise – boolean; whether this link's probability should be taken into account when normalising probabilities for alternatives in the confnet

UtteranceConfusionNetwork.**add**(*hyps*)
Adds a new arc to the confnet with alternatives as specified.

Arguments:

- **hyps: an iterable of simple hypotheses – (probability, word)** tuples

UtteranceConfusionNetwork.**cn**

UtteranceConfusionNetwork.**find**(*phrase*, *start=0*, *end=None*)

UtteranceConfusionNetwork.**find_unaware**(*phrase*, *start=0*, *end=None*)

UtteranceConfusionNetwork.**get_best_hyp**()

UtteranceConfusionNetwork.**get_best_utterance**()

UtteranceConfusionNetwork.**get_hyp_index_utterance**(*hyp_index*)

UtteranceConfusionNetwork.**get_next_worse_candidates**(*hyp_index*)
Returns such hypotheses that will have lower probability. It assumes that the confusion network is sorted.

UtteranceConfusionNetwork.**get_phrase_idxs**(*phrase*, *start=0*, *end=None*, *start_in_midlinks=True*, *immediate=False*)
Returns indices to words constituting the given phrase within this confnet. It looks only for the first occurrence of the phrase in the interval specified.

Arguments: phrase: the phrase to look for, specified as a list of words start: the index where to start searching end: the index after which to stop searching start_in_midlinks: whether a phrase starting in the middle of

a long link should be considered too

immediate: whether the phrase has to start immediately at the start index (intervening empty words are allowed)

Returns:

- an empty list in case that phrase was not found
- a list of indices to words (UtteranceConfusionNetwork.Index) that constitute that phrase within this confnet

UtteranceConfusionNetwork.**get_prob**(*hyp_index*)
Returns a probability of the given hypothesis.

UtteranceConfusionNetwork.**get_utterance_nblist**(*n=10*, *prune_prob=0.005*)
Parses the confusion network and generates n best hypotheses.

The result is a list of utterance hypotheses each with a with assigned probability. The list also includes the utterance "_other_" for not having the correct utterance in the list.

Generation of hypotheses will stop when the probability of the hypotheses is smaller then the `prune_prob`.

UtteranceConfusionNetwork.**index**(*phrase*, *start=0*, *end=None*)

UtteranceConfusionNetwork.**isempty**()

UtteranceConfusionNetwork.**iter_ngrams**(*n*, *with_boundaries=False*, *start=None*)
Iterates n-gram hypotheses of the length specified. This is the interface method. It is aware of multi-word phrases ("long links") that were substituted into the confnet.

Arguments: n: size of the n-grams with_boundaries: whether to include special sentence boundary marks start: at which word index the n-grams have to start (exactly)

UtteranceConfusionNetwork.**iter_ngrams_fromto**(*from_=None*, *to=None*)
Iterates n-gram hypotheses between the indices **'from_'** and **'to_'**. This method does not consider phrases longer than 1 that were substituted into the confnet.

UtteranceConfusionNetwork.**iter_ngrams_unaware**(*n*, *with_boundaries=False*)
Iterates n-gram hypotheses of the length specified. This is the interface method, and uses 'iter_ngrams_fromto' internally. This method does not consider phrases longer than 1 that were substituted into the confnet.

Arguments: n: size of the n-grams with_boundaries: whether to include special sentence boundary marks

UtteranceConfusionNetwork.**iter_triples**()

UtteranceConfusionNetwork.**iter_typeval**()

UtteranceConfusionNetwork.**join_typeval**(*type_*, *val*)

UtteranceConfusionNetwork.**lower**()
Lowercases words of this confnet.

BEWARE, this method is destructive, it lowercases self.

classmethod UtteranceConfusionNetwork.**make_other**(*type_*)

UtteranceConfusionNetwork.**merge**()
Adds up probabilities for the same hypotheses.

TODO: not implemented yet

UtteranceConfusionNetwork.**normalise**(*end=None*)
Makes sure that all probabilities add up to one. There should be no need of calling this from outside, since this invariant is ensured between calls to this class' methods.

UtteranceConfusionNetwork.**other_val = (u'[OTHER]',)**

UtteranceConfusionNetwork.**phrase2category_label**(*phrase*, *catlab*)
Replaces the phrase given by 'phrase' by a new phrase, given by 'catlab'. Assumes 'catlab' is an abstraction for 'phrase'.

UtteranceConfusionNetwork.**prune**(*prune_prob=0.001*)

UtteranceConfusionNetwork.**replace**(*phrase*, *replacement*)

UtteranceConfusionNetwork.**replace_typeval**(*combined*, *replacement*)

UtteranceConfusionNetwork.**repr_escer = <alex.utils.text.Escaper object>**

UtteranceConfusionNetwork.**repr_spec_chars = u'():,;|[]"\\'**

UtteranceConfusionNetwork.**sort**()
Sort the alternatives for each word according their probability.

UtteranceConfusionNetwork.**str_escer = <alex.utils.text.Escaper object>**

exception alex.components.asr.utterance.**UtteranceConfusionNetworkException**
Bases: *alex.components.slu.exceptions.SLUException*

class alex.components.asr.utterance.**UtteranceConfusionNetworkFeatures**(*type=u'ngram'*, *size=3*, *confnet=None*)
Bases: *alex.ml.features.Features*

Represents features extracted from an utterance hypothesis in the form of a confusion network. These are simply a probabilistic generalisation of simple utterance features. Only n-gram (incl. skip n-gram) features are currently implemented.

parse(*confnet*)
Extracts the features from 'confnet'.

exception alex.components.asr.utterance.**UtteranceException**
Bases: *alex.components.slu.exceptions.SLUException*

class alex.components.asr.utterance.**UtteranceFeatures**(*type=u'ngram'*, *size=3*, *utterance=None*)
Bases: *alex.ml.features.Features*

Represents the vector of features for an utterance.

The class also provides methods for manipulation of the feature vector, including extracting features from an utterance.

Currently, only n-gram (including skip n-grams) features are implemented.

Attributes: type: type of features ('ngram') size: size of features (an integer) features: mapping { feature : value of feature (# occs) }

parse(*utterance*, *with_boundaries=True*)
Extracts the features from 'utterance'.

class alex.components.asr.utterance.**UtteranceHyp**(*prob=None*, *utterance=None*)
Bases: *alex.components.asr.utterance.ASRHypothesis*

Provide an interface for 1-best hypothesis from the ASR component.

get_best_utterance()

class alex.components.asr.utterance.**UtteranceNBList**(*rep=None*)
Bases: *alex.components.asr.utterance.ASRHypothesis*, *alex.ml.hypothesis.NBList*

Provides functionality of n-best lists for utterances.

When updating the n-best list, one should do the following.

1. add utterances or parse a confusion network
2. merge and normalise, in either order

Attributes:

n_best: the list containing pairs [prob, utterance] sorted from the most probable to the least probable ones

add_other()

deserialise(*rep*)

get_best()

get_best_utterance()
Returns the most probable utterance.

DEPRECATED. Use get_best instead.

normalise()
The N-best list is extended to include the "_other_" utterance to represent those utterance hypotheses which are not included in the N-best list.

DEPRECATED. Use add_other instead.

scale()
Scales the n-best list to sum to one.

serialise()

sort()
DEPRECATED, going to be removed.

exception alex.components.asr.utterance.**UtteranceNBListException**
Bases: *alex.components.slu.exceptions.SLUException*

class alex.components.asr.utterance.**UtteranceNBListFeatures**(*type=u'ngram'*, *size=3*, *utt_nblist=None*)
Bases: *alex.ml.features.Features*

parse(*utt_nblist*)
This should be called only once during the object's lifetime, preferably from within the initialiser.

alex.components.asr.utterance.**load_utt_confnets**(*fname*, *limit=None*, *encoding=u'UTF-8'*)
Loads a dictionary of utterance confusion networks from a given file.

The file is assumed to contain lines of the following form:

[whitespace..]<key>[whitespace..]=>[whitespace..]<utt_cn>[whitespace..]

or just (without keys):

[whitespace..]<utt_cn>[whitespace..]

where <utt_cn> is obtained as repr() of an UtteranceConfusionNetwork object.

Arguments:

fname – path towards the file to read the utterance confusion networks from

limit – limit on the number of confusion networks to read encoding – the file encoding

Returns a dictionary with confnets (instances of Utterance) as values.

alex.components.asr.utterance.**load_utt_nblists**(*fname*, *limit=None*, *n=40*, *encoding=u'UTF-8'*)
Loads a dictionary of utterance n-best lists from a file with confnets.

The n-best lists are obtained simply from the confnets.

The file is assumed to contain lines of the following form:

[whitespace..]<key>[whitespace..]=>[whitespace..]<utt_cn>[whitespace..]

or just (without keys):

[whitespace..]<utt_cn>[whitespace..]

where <utt_cn> is obtained as repr() of an UtteranceConfusionNetwork object.

Arguments:

fname – path towards the file to read the utterance confusion networks from

limit – limit on the number of n-best lists to read n – depth of n-best lists encoding – the file encoding

Returns a dictionary with n-best lists (instances of UtteranceNBList) as values.

alex.components.asr.utterance.**load_utterances**(*fname*, *limit=None*, *encoding=u'UTF-8'*)
Loads a dictionary of utterances from a given file.

The file is assumed to contain lines of the following form:

[whitespace..]<key>[whitespace..]=>[whitespace..]<utterance>[whitespace..]

or just (without keys):

[whitespace..]<utterance>[whitespace..]

Arguments: fname – path towards the file to read the utterances from limit – limit on the number of utterances to read encoding – the file encoding

Returns a dictionary with utterances (instances of Utterance) as values.

`alex.components.asr.utterance.`**`save_utterances`**(*file_name*, *utt*, *encoding=u'UTF-8'*)

Saves a dictionary of utterances in the wave as key format into a file.

Parameters

- **`file_name`** – name of the target file
- **`utt`** – a dictionary with the utterances where the keys are the names of the corresponding wave files

Returns None

Module contents

alex.components.dm package

Submodules

alex.components.dm.autopath module self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing "pypy" is in sys.path.

If you modify the master "autopath.py" version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

alex.components.dm.base module

class `alex.components.dm.base.`**`DialogueManager`**(*cfg*)

Bases: `object`

This is a base class for a dialogue manager. The purpose of a dialogue manager is to accept input in the form dialogue acts and respond again in the form of dialogue acts.

The dialogue manager should be able to accept multiple inputs without producing any output and be able to produce multiple outputs without any input.

`da_in`(*da*, *utterance=None*)

Receives an input dialogue act or dialogue act list with probabilities or dialogue act confusion network.

When the dialogue act is received an update of the state is performed.

da_out()
Produces output dialogue act.

end_dialogue()
Ends the dialogue and post-process the data.

log_state()
Log the state of the dialogue state.

Returns none

new_dialogue()
Initialises the dialogue manager and makes it ready for a new dialogue conversation.

class alex.components.dm.base.**DialoguePolicy**(*cfg*, *ontology*)
Bases: object

This is a base class policy.

get_da(*dialogue_state*)

class alex.components.dm.base.**DialogueState**(*cfg*, *ontology*)
Bases: object

This is a trivial implementation of a dialogue state and its update.

It uses only the best dialogue act from the input and based on this it updates its state.

get_slots_being_confirmed()
Returns all slots which are currently being confirmed by the user along with the value being confirmed.

get_slots_being_noninformed()
Returns all slots provided by the user and the system has not informed about them yet along with the value of the slot.

get_slots_being_requested()
Returns all slots which are currently being requested by the user along with the correct value.

log_state()
Log the state using the the session logger.

restart()
Reinitialises the dialogue state so that the dialogue manager can start from scratch.

Nevertheless, remember the turn history.

update(*user_da*, *system_da*)
Interface for the dialogue act update.

It can process dialogue act, dialogue act N best lists, or dialogue act confusion networks.

Parameters

- **user_da** (*DialogueAct*, *DialogueActNBList* or *DialogueActConfusionNetwork*) – Dialogue act to process.
- **system_da** – Last system dialogue act.

class alex.components.dm.base.**DiscreteValue**(*values*, *name=''*, *desc=''*)
Bases: object

explain(*full=False*, *linear_prob=False*)
This function prints the values and their probabilities for this node.

mph()
The function returns the most probable value and its probability in a tuple.

mpv()
The function returns the most probable value.

mpvp()
The function returns the probability of the most probable value.

normalise()
This function normalise the sum of all probabilities to 1.0

prune(*threshold=0.001*)
Prune all values with probability less then a threshold.

tmphs()
This function returns two most probable values and their probabilities.

The function returns a tuple consisting of two tuples (probability, value).

tmpvs()
The function returns two most probable values.

tmpvsp()
The function returns probabilities of two most probable values in the slot.

alex.components.dm.common module

alex.components.dm.common.**dm_factory**(*dm_type*, *cfg*)
alex.components.dm.common.**get_dm_type**(*cfg*)

alex.components.dm.dddstate module

class alex.components.dm.dddstate.**D3DiscreteValue**(*values={}*, *name=u''*, *desc=u''*)
Bases: *alex.components.dm.base.DiscreteValue*

This is a simple implementation of a probabilistic slot. It serves for the case of simple MDP approach or UFAL DSTC 1.0-like dialogue state deterministic update.

add(*value*, *prob*)
This function adds probability to the given value.

distribute(*value*, *dist_prob*)
This function distributes a portion of probability mass assigned to the `value` to other values with a weight `prob`.

explain(*full=False*, *linear_prob=True*)
This function prints the values and their probabilities for this node.

get(*value*, *default_prob*)

items()

mph()
The function returns the most probable value and its probability in a tuple.

normalise()
This function normalises the sum of all probabilities to 1.0

reset()

scale(*weight*)
This function scales each probability by the weigh.t

set(*value*, *prob=None*)
This function sets a probability of a specific value.

WARNING This can lead to un-normalised probabilities.

test(*test_value=None*, *test_prob=None*, *neg_val=False*, *neg_prob=False*)
Test the most probable value of the slot whether:

1.the most probable value is equal to test_value and

2.its probability is larger the test_prob

Each of the above tests can be negated when neg_* is set True.

Parameters

- **test_value** –
- **test_prob** –
- **neg_val** –
- **neg_prob** –

Returns

tmphs()
This function returns two most probable values and their probabilities. If there are multiple values with the same probability, it prefers non-'none' values.

The function returns a tuple consisting of two tuples (probability, value).

Return type tuple

class alex.components.dm.dddstate.**DeterministicDiscriminativeDialogueState**(*cfg*, *ontology*)
Bases: *alex.components.dm.base.DialogueState*

This is a trivial implementation of a dialogue state and its update.

It uses only the best dialogue act from the input. Based on this it updates its state.

get_accepted_slots(*acc_prob*)
Returns all slots which have a probability of a non "none" value larger then some threshold.

get_changed_slots(*cha_prob*)
Returns all slots that has changed from the previous turn. Because the change is determined by change in probability for a particular value, there may be very small changes. Therefore, this will only report changes for values with a probability larger than the given threshold.

Parameters **cha_prob** – minimum current probability of the most probable hypothesis to be reported

Return type dict

get_slots_being_confirmed(*conf_prob=0.8*)
Return all slots which are currently being confirmed by the user along with the value being confirmed.

get_slots_being_noninformed(*noninf_prob=0.8*)
Return all slots provided by the user and the system has not informed about them yet along with the value of the slot.

This will not detect a change in a goal. For example:

```
U: I want a Chinese restaurant.
S: Ok, you want a Chinese restaurant. What price range you have in mind?
U: Well, I would rather want an Italian Restaurant.
S: Ok, no problem. You want an Italian restaurant. What price range you have in mind?
```

Because the system informed about the food type and stored "system-informed", then we will not notice that we confirmed a different food type.

get_slots_being_requested(*req_prob=0.8*)
Return all slots which are currently being requested by the user along with the correct value.

get_slots_tobe_confirmed(*min_prob*, *max_prob*)
Returns all slots which have a probability of a non "none" value larger then some threshold and still not so large to be considered as accepted.

get_slots_tobe_selected(*sel_prob*)
Returns all slots which have a probability of the two most probable non "none" value larger then some threshold.

has_state_changed(*cha_prob*)
Returns a boolean indicating whether the dialogue state changed significantly since the last turn. True is returned if at least one slot has at least one value whose probability has changed at least by the given threshold since last time.

Parameters **cha_prob** – minimum probability change to be reported

Return type Boolean

log_state()
Log the state using the the session logger.

restart()
Reinitialise the dialogue state so that the dialogue manager can start from scratch.

Nevertheless, remember the turn history.

slots = None

update(*user_da*, *system_da*)
Interface for the dialogue act update.

It can process dialogue act, dialogue act N best lists, or dialogue act confusion networks.

Parameters

- **user_da** (*DialogueAct*, *DialogueActNBList* or *DialogueActConfusionNetwork*) – Dialogue act to process.
- **system_da** – Last system dialogue act.

alex.components.dm.dstc_tracker module

class alex.components.dm.dstc_tracker.**DSTCState**(*slots*)
Bases: object

Represents state of the tracker.

pprint()
Pretty-print self.

class alex.components.dm.dstc_tracker.**DSTCTracker**(*slots*, *default_space_size=defaultdict(<function <lambda> at 0x7fbf78f57668>, {})*)
Bases: *alex.components.dm.tracker.StateTracker*

Represents simple deterministic DSTC state tracker.

state_class
alias of DSTCState

update_state(*state*, *cn*)

class alex.components.dm.dstc_tracker.**ExtendedSlotUpdater**
Bases: object

Updater of state given observation and deny distributions.

classmethod update_slot(*curr_pd*, *observ_pd*, *deny_pd*)

alex.components.dm.dstc_tracker.**main**()

alex.components.dm.dummypolicy module This is an example implementation of a dummy yet funny dialogue policy.

class alex.components.dm.dummypolicy.**DummyDialoguePolicy**(*cfg*, *ontology*)
Bases: *alex.components.dm.base.DialoguePolicy*

This is a trivial policy just to demonstrate basic functionality of a proper DM.

get_da(*dialogue_state*)

alex.components.dm.exceptions module
exception alex.components.dm.exceptions.**DMException**
Bases: *alex.AlexException*
exception alex.components.dm.exceptions.**DeterministicDiscriminativeDialogueStateException**
Bases: *alex.components.dm.exceptions.DialogueStateException*

exception alex.components.dm.exceptions.**DialogueManagerException**
Bases: *alex.AlexException*

exception alex.components.dm.exceptions.**DialoguePolicyException**
Bases: *alex.AlexException*

exception alex.components.dm.exceptions.**DialogueStateException**
Bases: *alex.AlexException*

exception alex.components.dm.exceptions.**DummyDialoguePolicyException**
Bases: *alex.components.dm.exceptions.DialoguePolicyException*

alex.components.dm.ontology module
class alex.components.dm.ontology.**Ontology**(*file_name=None*)
Bases: object

Represents an ontology for a dialogue domain.

get_compatible_vals(*slot_pair*, *value*)
Given a slot pair (key to 'compatible_values' in ontology data), this returns the set of compatible values for the given key. If there is no information about the given pair, None is returned.

Parameters

- **slot_pair** – key to 'compatible_values' in ontology data
- **value** – the subkey to check compatible values for

Return type *set*

get_default_value(*slot*)
Given a slot name, get its default value (if set in the ontology). Returns None if the default value is not set for the given slot.

Parameters **slot** – the name of the desired slot

Return type unicode

is_compatible(*slot_pair*, *val1*, *val2*)
Given a slot pair and a pair of values, this tests whether the values are compatible. If there is no information about the slot pair or the first value, returns False. If the second value is None, returns always True (i.e. None is compatible with anything).

Parameters

- **slot_pair** – key to 'compatible_values' in ontology data
- **val1** – value of the 1st slot
- **val2** – value of the 2nd slot

Return type Boolean

last_talked_about(**args*, ***kwds*)
Returns a list of slots and values that should be used to for tracking about what was talked about recently, given the input dialogue acts.

Parameters

- **da_type** – the source dialogue act type
- **name** – the source slot name
- **value** – the source slot value

Returns returns a list of target slot names and values used for tracking

load(*file_name*)

reset_on_change(**args*, ***kwds*)

slot_has_value(*name*, *value*)
Check whether the slot and the value are compatible.

slot_is_binary(*name*)
Check whether the given slot has a binary value (using the 'binary' key in the 'slot_attributes' for the given slot name).

Parameters **name** – name of the slot being checked

slots_system_confirms(**args*, ***kwds*)
Return all slots the system can request.

slots_system_requests(**args*, ***kwds*)
Return all slots the system can request.

slots_system_selects(**args*, ***kwds*)
Return all slots the system can request.

exception alex.components.dm.ontology.**OntologyException**
Bases: exceptions.Exception

alex.components.dm.pstate module

class alex.components.dm.pstate.**PDDiscrete**(*initial=None*)
Bases: *alex.components.dm.pstate.PDDiscreteBase*

Discrete probability distribution.

NULL = None

OTHER = '<other>'

get(*item*)

get_distrib()

get_entropy()

get_items()

iteritems()

meta_slots = set([None, '<other>'])

normalize()
Normalize the probability distribution.

update(*items*)

class alex.components.dm.pstate.**PDDiscreteBase**(**args*, ***kwargs*)
Bases: object

get_best()

get_max(*which_one=0*)

remove(*item*)

class alex.components.dm.pstate.**PDDiscreteOther**(*space_size*, *initial=None*)
Bases: *alex.components.dm.pstate.PDDiscreteBase*

Discrete probability distribution with sink probability slot for OTHER.

NULL = None

OTHER = '<other>'

get(*item*)

get_distrib()

get_entropy()

get_items()

get_max(*which_one=0*)

iteritems()

meta_slots = set([None, '<other>'])

normalize(*redistrib=0.0*)
Normalize the probability distribution.

space_size = None

update(*items*)

class alex.components.dm.pstate.**SimpleUpdater**(*slots*)
Bases: object

update(*observ*)

update_slot(*slot*, *observ_distrib*)

alex.components.dm.state module

class alex.components.dm.state.**State**(*slots*)

Bases: object

update(*item*, *value*)

alex.components.dm.tracker module

class alex.components.dm.tracker.**StateTracker**

Bases: object

state_class = None

update_state(*state*, *cn*)

Update state according to the confusion network cn.

Module contents

alex.components.hub package

Submodules

alex.components.hub.ahub module

alex.components.hub.aio module

alex.components.hub.asr module

class alex.components.hub.asr.**ASR**(*cfg*, *commands*, *audio_in*, *asr_hypotheses_out*, *close_event*)

Bases: multiprocessing.process.Process

ASR recognizes input audio and returns an N-best list hypothesis or a confusion network.

Recognition starts with the "speech_start()" command in the input audio stream and ends with the "speech_end()" command.

When the "speech_end()" command is received, the component asks responsible ASR module to return hypotheses and sends them to the output.

This component is a wrapper around multiple recognition engines which handles inter-process communication.

Attributes: asr – the ASR object itself

process_pending_commands()

Process all pending commands.

Available commands: stop() - stop processing and exit the process flush() - flush input buffers.

Now it only flushes the input connection.

Returns True iff the process should terminate.

read_audio_write_asr_hypotheses()

recv_input_locally()
Copy all input from input connections into local queue objects.

This will prevent blocking the senders.

run()

alex.components.hub.calldb module

class alex.components.hub.calldb.**CallDB**(*cfg*, *file_name*, *period=86400*)
Bases: object

Implements logging of all interesting call stats. It can be used for customization of the SDS, e.g. for novice or expert users.

close_database(*db*)

get_uri_stats(*remote_uri*)

log()

log_uri(*remote_uri*)

open_database()

read_database()

release_database()

track_confirmed_call(*remote_uri*)

track_disconnected_call(*remote_uri*)

alex.components.hub.dm module

class alex.components.hub.dm.**DM**(*cfg*, *commands*, *slu_hypotheses_in*, *dialogue_act_out*, *close_event*)
Bases: multiprocessing.process.Process

DM accepts N-best list hypothesis or a confusion network generated by an SLU component. The result of this component is an output dialogue act.

When the component receives an SLU hypothesis then it immediately responds with an dialogue act.

This component is a wrapper around multiple dialogue managers which handles multiprocessing communication.

epilogue()
Gives the user last information before hanging up.

:return the name of the activity or None

epilogue_final_apology()

epilogue_final_code()

epilogue_final_question()

process_pending_commands()
Process all pending commands.

Available commands: stop() - stop processing and exit the process flush() - flush input buffers.

Now it only flushes the input connection.

Return True if the process should terminate.

read_slu_hypotheses_write_dialogue_act()

run()

test_code_server_connection()
this opens a test connection to our code server, content of the response is not important if our server is down this call will fail and the vm will crash. this is more sensible to CF people, otherwise CF contributor would do the job without getting paid.

alex.components.hub.exceptions module

exception alex.components.hub.exceptions.**VoipIOException**
Bases: *alex.AlexException*

alex.components.hub.hub module

class alex.components.hub.hub.**Hub**(*cfg*)
Bases: object

Common functionality for the hubs.

hub_type = 'Hub'

init_readline()
Initialize the readline functionality to enable console history.

write_readline()

alex.components.hub.messages module

class alex.components.hub.messages.**ASRHyp**(*hyp*, *source=None*, *target=None*, *fname=None*)
Bases: *alex.components.hub.messages.Message*

class alex.components.hub.messages.**Command**(*command*, *source=None*, *target=None*)
Bases: *alex.components.hub.messages.Message*

class alex.components.hub.messages.**DMDA**(*da*, *source=None*, *target=None*)
Bases: *alex.components.hub.messages.Message*

class alex.components.hub.messages.**Frame**(*payload*, *source=None*, *target=None*)
Bases: *alex.components.hub.messages.Message*

class alex.components.hub.messages.**Message**(*source*, *target*)
Bases: *alex.utils.mproc.InstanceID*

Abstract class which implements basic functionality for messages passed between components in the alex.

get_time_str()
Return current time in dashed ISO-like format.

class alex.components.hub.messages.**SLUHyp**(*hyp*, *asr_hyp=None*, *source=None*, *target=None*)
Bases: *alex.components.hub.messages.Message*

class alex.components.hub.messages.**TTSText**(*text*, *source=None*, *target=None*)
Bases: *alex.components.hub.messages.Message*

alex.components.hub.nlg module

class alex.components.hub.nlg.**NLG**(*cfg*, *commands*, *dialogue_act_in*, *text_out*, *close_event*)
Bases: multiprocessing.process.Process

The NLG component receives a dialogue act generated by the dialogue manager and then it converts the act into the text.

This component is a wrapper around multiple NLG components which handles multiprocessing communication.

process_da(*da*)

process_pending_commands()
Process all pending commands.

Available commands: stop() - stop processing and exit the process flush() - flush input buffers.

Now it only flushes the input connection.

Return True if the process should terminate.

read_dialogue_act_write_text()

run()

alex.components.hub.slu module

class alex.components.hub.slu.**SLU**(*cfg*, *commands*, *asr_hypotheses_in*, *slu_hypotheses_out*, *close_event*)
Bases: multiprocessing.process.Process

The SLU component receives ASR hypotheses and converts them into hypotheses about the meaning of the input in the form of dialogue acts.

This component is a wrapper around multiple SLU components which handles inter-process communication.

process_pending_commands()
Process all pending commands.

Available commands: stop() - stop processing and exit the process flush() - flush input buffers.

Now it only flushes the input connection.

Return True if the process should terminate.

read_asr_hypotheses_write_slu_hypotheses()

run()

alex.components.hub.tts module

alex.components.hub.vad module

alex.components.hub.vio module

alex.components.hub.webio module

Module contents

alex.components.nlg package

Subpackages

alex.components.nlg.tectotpl package

Subpackages

alex.components.nlg.tectotpl.block package

Subpackages

alex.components.nlg.tectotpl.block.a2w package

Subpackages

alex.components.nlg.tectotpl.block.a2w.cs package

Submodules

alex.components.nlg.tectotpl.block.a2w.cs.concatenatetokens module

class alex.components.nlg.tectotpl.block.a2w.cs.concatenatetokens.**ConcatenateTokens**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.core.block.Block*

Detokenize the sentence, spread whitespace correctly.

process_zone(*zone*)

Detokenize the sentence and assign the result to the sentence attribute of the current zone.

alex.components.nlg.tectotpl.block.a2w.cs.removerepeatedtokens module

class alex.components.nlg.tectotpl.block.a2w.cs.removerepeatedtokens.**RemoveRepeatedTokens**(*scen*
arg

Bases: *alex.components.nlg.tectotpl.core.block.Block*

Remove two identical neighboring tokens.

process_zone(*zone*)

Remove two identical neighboring tokens in the given sentence.

Module contents

Module contents

alex.components.nlg.tectotpl.block.read package

Submodules

alex.components.nlg.tectotpl.block.read.tectotemplates module

class alex.components.nlg.tectotpl.block.read.tectotemplates.**TectoTemplates**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.core.block.Block*

Reader for partial t-tree dialog system templates, where treelets can be intermixed with linear text.

Example template:

Vlak přijede v [[7|adj:attr] hodina|n:4|gender:fem].

All linear text is inserted into t-lemmas of atomic nodes, while treelets have their formeme and grammateme values filled in.

parse_line(*text*, *troot*)
Parse a template to a t-tree.

parse_treelet(*text*, *tnode*)
Parse a treelet in the template, filling the required values. Returns the position in the text after the treelet.

process_document(*filename*)
Read a Tecto-Template file and return its contents as a Document object.

alex.components.nlg.tectotpl.block.read.yaml module

class alex.components.nlg.tectotpl.block.read.yaml.**YAML**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.core.block.Block*

process_document(*filename*)
Read a YAML file and return its contents as a Document object

Module contents

alex.components.nlg.tectotpl.block.t2a package

Subpackages

alex.components.nlg.tectotpl.block.t2a.cs package

Submodules

alex.components.nlg.tectotpl.block.t2a.cs.addappositionpunct module

class alex.components.nlg.tectotpl.block.t2a.cs.addappositionpunct.**AddAppositionPunct**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.core.block.Block*

Separating Czech appositions, such as in 'John, my best friend, ...' with commas.

Arguments: language: the language of the target tree selector: the selector of the target tree

add_comma_node(*aparent*)
Add a comma a-node to the given parent

is_before_punct(*anode*)
Test whether the subtree of the given node precedes a punctuation node.

process_tnode(*tnode*)
Adds punctuation a-nodes if the given node is an apposition node.

alex.components.nlg.tectotpl.block.t2a.cs.addauxverbcompoundfuture module

class alex.components.nlg.tectotpl.block.t2a.cs.addauxverbcompoundfuture.**AddAuxVerbCompoundFu**

Bases: *alex.components.nlg.tectotpl.core.block.Block*

Add compound future auxiliary 'bude'.

Arguments: language: the language of the target tree selector: the selector of the target tree

process_tnode(*tnode*)
Add compound future auxiliary to a node, where appropriate.

alex.components.nlg.tectotpl.block.t2a.cs.addauxverbcompoundpassive module

class alex.components.nlg.tectotpl.block.t2a.cs.addauxverbcompoundpassive.**AddAuxVerbCompoundP**

Bases: *alex.components.nlg.tectotpl.core.block.Block*

Add compound passive auxiliary 'být'.

Arguments: language: the language of the target tree selector: the selector of the target tree

process_tnode(*tnode*)
Add compound passive auxiliary to a node, where appropriate.

alex.components.nlg.tectotpl.block.t2a.cs.addauxverbcompoundpast module

class alex.components.nlg.tectotpl.block.t2a.cs.addauxverbcompoundpast.**AddAuxVerbCompoundPast**

Bases: *alex.components.nlg.tectotpl.core.block.Block*

Add compound past tense auxiliary of the 1st and 2nd person 'jsem/jsi/jsme/jste'.

Arguments: language: the language of the target tree selector: the selector of the target tree

AUX_PAST_FORMS = {(u'P', u'2'): u'jste', (u'S', u'1'): u'jsem', (u'S', u'2'): u'jsi', (u'.', u'2'): u'jsi', (u'P', u'1'): u'jsme'

process_tnode(*tnode*)
Add compound past auxiliary to a node, where appropriate.

alex.components.nlg.tectotpl.block.t2a.cs.addauxverbconditional module

class alex.components.nlg.tectotpl.block.t2a.cs.addauxverbconditional.**AddAuxVerbConditional**(*s*
a

Bases: *alex.components.nlg.tectotpl.core.block.Block*

Add conditional auxiliary 'by'/'bych'.

Arguments: language: the language of the target tree selector: the selector of the target tree

process_tnode(*tnode*)
Add conditional auxiliary to a node, where appropriate.

alex.components.nlg.tectotpl.block.t2a.cs.addauxverbmodal module

class alex.components.nlg.tectotpl.block.t2a.cs.addauxverbmodal.**AddAuxVerbModal**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.core.block.Block*

Add modal verbs.

Arguments: language: the language of the target tree selector: the selector of the target tree

DEONTMOD_2_MODAL = {u'vol': u'cht\xedt', u'hrt': u'm\xedt', u'perm': u'moci', u'fac': u'moci', u'deb': u'muset', u'po

process_tnode(*tnode*)
Add modal auxiliary to a node, where appropriate.

alex.components.nlg.tectotpl.block.t2a.cs.addclausalexpletives module

class alex.components.nlg.tectotpl.block.t2a.cs.addclausalexpletives.**AddClausalExpletives**(*scen*, *args*)

Bases: *alex.components.nlg.tectotpl.block.t2a.addauxwords.AddAuxWords*

Add clausal expletive pronoun 'to' (+preposition) to subordinate clauses with 'že', if the parent verb requires it.

Arguments: language: the language of the target tree selector: the selector of the target tree

get_anode(*tnode*)
Return the a-node that is the root of the verbal a-subtree.

get_aux_forms(*tnode*)
Return the clausal expletive to be added, if supposed to.

new_aux_node(*anode*, *form*)
Create a node for the expletive/its preposition.

postprocess(*tnode*, *anode*, *aux_anodes*)
Rehang the conjunction 'že', now above the expletive, under it. Fix clause numbers and ordering.

alex.components.nlg.tectotpl.block.t2a.cs.addclausalpunct module

class alex.components.nlg.tectotpl.block.t2a.cs.addclausalpunct.**AddClausalPunct**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.core.block.Block*

An abstract ancestor for blocks working with clausal punctuation.

Arguments: language: the language of the target tree selector: the selector of the target tree

is_clause_in_quotes(*anode*)
Return True if the given node is in an enquoted clause. The node must end the clause.

alex.components.nlg.tectotpl.block.t2a.cs.addcoordpunct module

class alex.components.nlg.tectotpl.block.t2a.cs.addcoordpunct.**AddCoordPunct**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.core.block.Block*

Add comma to coordinated lists of 3 and more elements, as well as before some Czech coordination conjunctions ('ale', 'ani').

Arguments: language: the language of the target tree selector: the selector of the target tree

add_comma_node(*anode*)
Add a comma AuxX node under the given node.

is_at_clause_boundary(*anode*)
Return true if the given node is at a clause boundary (i.e. the nodes immediately before and after it belong to different clauses).

process_anode(*anode*)
Add coordination punctuation to the given anode, if applicable.

alex.components.nlg.tectotpl.block.t2a.cs.addparentheses module

class alex.components.nlg.tectotpl.block.t2a.cs.addparentheses.**AddParentheses**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.core.block.Block*

Add '(' / ')' nodes to nodes which have the wild/is_parenthesis attribute set.

Arguments: language: the language of the target tree selector: the selector of the target tree

add_parenthesis_node(*anode*, *lemma*, *clause_num*)
Add a parenthesis node as a child of the specified a-node; with the given lemma and clause number set.

continued_paren_left(*anode*)
Return True if this node is continuing a parenthesis from the left.

continued_paren_right(*anode*)
Return True if a parenthesis continues after this node to the right.

process_anode(*anode*)
Add parentheses to an a-node, where appropriate.

alex.components.nlg.tectotpl.block.t2a.cs.addprepositions module

class alex.components.nlg.tectotpl.block.t2a.cs.addprepositions.**AddPrepositions**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.block.t2a.addauxwords.AddAuxWords*

Add prepositional a-nodes according to formemes.

Arguments: language: the language of the target tree selector: the selector of the target tree

get_aux_forms(*tnode*)
Find prepositional nodes to be created.

new_aux_node(*anode*, *form*)
Create a prepositional node with the given preposition form and parent.

postprocess(*tnode*, *anode*, *aux_nodes*)
Move rhematizers in front of the newly created PPs.

alex.components.nlg.tectotpl.block.t2a.cs.addreflexiveparticles module

class alex.components.nlg.tectotpl.block.t2a.cs.addreflexiveparticles.**AddReflexiveParticles**(*s*
a

Bases: *alex.components.nlg.tectotpl.core.block.Block*

Add reflexive particles to reflexiva tantum and reflexive passive verbs.

Arguments: language: the language of the target tree selector: the selector of the target tree

process_tnode(*tnode*)
Add reflexive particle to a node, if applicable.

alex.components.nlg.tectotpl.block.t2a.cs.addsentfinalpunct module

class alex.components.nlg.tectotpl.block.t2a.cs.addsentfinalpunct.**AddSentFinalPunct**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.block.t2a.cs.addclausalpunct.AddClausalPunct*

Add final sentence punctuation ('?', '.').

Arguments: language: the language of the target tree selector: the selector of the target tree

process_ttree(*troot*)
Add final punctuation to the given sentence.

alex.components.nlg.tectotpl.block.t2a.cs.addsubconjs module

class alex.components.nlg.tectotpl.block.t2a.cs.addsubconjs.**AddSubconjs**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.block.t2a.addauxwords.AddAuxWords*

Add subordinate conjunction a-nodes according to formemes.

Arguments: language: the language of the target tree selector: the selector of the target tree

get_aux_forms(*tnode*)
Find prepositional nodes to be created.

new_aux_node(*anode*, *form*)
Create a subordinate conjunction node with the given conjunction form and parent.

alex.components.nlg.tectotpl.block.t2a.cs.addsubordclausepunct module

class alex.components.nlg.tectotpl.block.t2a.cs.addsubordclausepunct.**AddSubordClausePunct**(*scen*, *args*)

Bases: *alex.components.nlg.tectotpl.block.t2a.cs.addclausalpunct.AddClausalPunct*

Add commas separating subordinate clauses.

Arguments: language: the language of the target tree selector: the selector of the target tree

are_in_coord_clauses(*aleft*, *aright*)
Check if the given nodes are in two coordinated clauses.

get_clause_parent(*anode*)
Return the parent of the clause the given node belongs to; the result may be the root of the tree.

insert_comma_between(*aleft*, *aright*)
Insert a comma node between these two nodes, find out where to hang it.

process_atree(*aroot*)
Add subordinate clause punctuation to the given sentence.

alex.components.nlg.tectotpl.block.t2a.cs.capitalizesentstart module

class alex.components.nlg.tectotpl.block.t2a.cs.capitalizesentstart.**CapitalizeSentStart**(*scenar*, *args*)

Bases: *alex.components.nlg.tectotpl.core.block.Block*

Capitalize the first word in the sentence (skip punctuation etc.).

OPEN_PUNCT = u'^[({[\u201a\u201e\xab\u2039|*"\\\']+$'

process_zone(*zone*)
Find the first valid word in the sentence and capitalize it.

alex.components.nlg.tectotpl.block.t2a.cs.deletesuperfluousauxs module

class alex.components.nlg.tectotpl.block.t2a.cs.deletesuperfluousauxs.**DeleteSuperfluousAuxs**(*s*, *a*

Bases: *alex.components.nlg.tectotpl.core.block.Block*

Delete repeated prepositions and and conjunctions in coordinations.

BASE_DIST_LIMIT = 8

DIST_LIMIT = {u'mezi': 50, u'pro': 8, u'proto\u017ee': 5, u'v': 5}

process_tnode(*tnode*)
Check for repeated prepositions and and conjunctions in coordinations, delete them if necessary.

alex.components.nlg.tectotpl.block.t2a.cs.dropsubjpersprons module

class alex.components.nlg.tectotpl.block.t2a.cs.dropsubjpersprons.**DropSubjPersProns**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.core.block.Block*

Remove the Czech pro-drop subject personal pronouns (or demonstrative "to") from the a-tree.

Arguments: language: the language of the target tree selector: the selector of the target tree

drop_anode(*tnode*)
Remove the lexical a-node corresponding to the given t-node

process_tnode(*tnode*)
Check if the a-node corresponding to the given t-node should be dropped, and do so where appropriate.

alex.components.nlg.tectotpl.block.t2a.cs.generatepossessiveadjectives module

class alex.components.nlg.tectotpl.block.t2a.cs.generatepossessiveadjectives.**GeneratePossessi**

Bases: *alex.components.nlg.tectotpl.core.block.Block*

According to formemes, this changes the lemma of the surface possessive adjectives from the original (deep) lemma which was identical to the noun from which the adjective is derived, e.g. changes the a-node lemma from 'Čapek' to 'Čapkův' if the corresponding t-node has the 'adj:poss' formeme.

Arguments: language: the language of the target tree selector: the selector of the target tree

load()

process_tnode(*tnode*)
Check a t-node if its lexical a-node should be changed; if yes, update its lemma.

alex.components.nlg.tectotpl.block.t2a.cs.generatewordforms module

class alex.components.nlg.tectotpl.block.t2a.cs.generatewordforms.**GenerateWordForms**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.core.block.Block*

Inflect word forms according to filled-in tags.

Arguments: language: the language of the target tree selector: the selector of the target tree

BACK_REGEX = <_sre.SRE_Pattern object>

load()
Load the model from a pickle.

process_atree(*aroot*)
Inflect word forms in the given a-tree.

alex.components.nlg.tectotpl.block.t2a.cs.imposeattragr module

class alex.components.nlg.tectotpl.block.t2a.cs.imposeattragr.**ImposeAttrAgr**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.block.t2a.imposeagreement.ImposeAgreement*

Impose case, gender and number agreement of attributes with their governing nouns.

Arguments: language: the language of the target tree selector: the selector of the target tree

impose(*tnode*, *match_nodes*)
Impose case, gender and number agreement on attributes.

process_excepts(*tnode*, *match_nodes*)
Handle special cases for this rule: nic/něco, numerals.

should_agree(*tnode*)
Find adjectives with a noun parent. Returns the a-layer nodes for the adjective and its parent, or False

alex.components.nlg.tectotpl.block.t2a.cs.imposecomplagr module

class alex.components.nlg.tectotpl.block.t2a.cs.imposecomplagr.**ImposeComplAgr**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.block.t2a.imposeagreement.ImposeAgreement*

Impose agreement of adjectival verb complements with the subject.

Arguments: language: the language of the target tree selector: the selector of the target tree

impose(*tnode*, *match_nodes*)
Impose the agreement on selected adjectival complements.

process_excepts(*tnode*, *match_nodes*)
Returns False; there are no special cases for this rule.

should_agree(*tnode*)
Find the complement and its subject.

alex.components.nlg.tectotpl.block.t2a.cs.imposepronzagr module

class alex.components.nlg.tectotpl.block.t2a.cs.imposepronzagr.**ImposePronZAgr**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.block.t2a.imposeagreement.ImposeAgreement*

In phrases such as 'každý z ...','žádná z ...', impose agreement in gender.

Arguments: language: the language of the target tree selector: the selector of the target tree

PRONOUNS = u'^(jeden|ka\u017ed\xfd|\u017e\xe1dn\xfd|oba|v\u0161echen|(n\u011b|lec)kter\xfd|(jak|kter)\xfdkoliv?|libov

impose(*tnode*, *tchild*)
Impose the gender agreement on selected nodes.

process_excepts(*tnode*, *match_nodes*)
Returns False; there are no special cases for this rule.

should_agree(*tnode*)
Find matching pronouns with 'z+2'-formeme children.

alex.components.nlg.tectotpl.block.t2a.cs.imposerelpronagr module

class alex.components.nlg.tectotpl.block.t2a.cs.imposerelpronagr.**ImposeRelPronAgr**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.block.t2a.imposeagreement.ImposeAgreement*

Impose gender and number agreement of relative pronouns with their antecedent.

Arguments: language: the language of the target tree selector: the selector of the target tree

impose(*tnode*, *tantec*)
Impose the gender agreement on selected nodes.

process_excepts(*tnode*, *match_nodes*)
Returns False; there are no special cases for this rule.

should_agree(*tnode*)
Find relative pronouns with a valid antecedent.

alex.components.nlg.tectotpl.block.t2a.cs.imposesubjpredagr module

class alex.components.nlg.tectotpl.block.t2a.cs.imposesubjpredagr.**ImposeSubjPredAgr**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.block.t2a.imposeagreement.ImposeAgreement*

Impose gender and number agreement of relative pronouns with their antecedent.

Arguments: language: the language of the target tree selector: the selector of the target tree

impose(*tnode*, *match_nodes*)
Impose the subject-predicate agreement on regular nodes.

process_excepts(*tnode*, *match_nodes*)
Returns False; there are no special cases for this rule.

should_agree(*tnode*)
Find finite verbs, with/without a subject.

alex.components.nlg.tectotpl.block.t2a.cs.initmorphcat module

class alex.components.nlg.tectotpl.block.t2a.cs.initmorphcat.**InitMorphcat**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.core.block.Block*

According to t-layer grammatemes, this initializes the morphcat structure at the a-layer that is the basis for a later POS tag limiting in the word form generation.

Arguments: language: the language of the target tree selector: the selector of the target tree

DEGREE = {u'comp': u'2', u'pos': u'1', u'acomp': u'2', u'sup': u'3', None: u'.', u'nr': u'.'}

GENDER = {u'anim': u'M', u'fem': u'F', u'inan': u'I', u'inher': u'.', u'neut': u'N', None: u'.', u'nr': u'.'}

NEGATION = {None: u'A', u'neg0': u'A', u'neg1': u'N'}

NUMBER = {None: u'.', u'nr': u'.', u'sg': u'S', u'pl': u'P', u'inher': u'.'}

PERSON = {u'1': u'1', None: u'.', u'3': u'3', u'2': u'2', u'inher': u'.'}

VOICE = {u'pas': u'P', u'deagent': u'A', u'passive': u'P', u'act': u'A', u'active': u'A', None: u'.'}

process_tnode(*tnode*)
Initialize the morphcat structure in the given node

set_case(*tnode*, *anode*)
Set the morphological case for an a-node according to the corresponding t-node's formeme, where applicable.

set_perspron_categories(*tnode*, *anode*)
Set detailed morphological categories of personal pronouns of various types (possessive, reflexive, personal per se)

alex.components.nlg.tectotpl.block.t2a.cs.marksubject module

class alex.components.nlg.tectotpl.block.t2a.cs.marksubject.**MarkSubject**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.core.block.Block*

Marks the subject of each clause with the Afun 'Sb'.

Arguments: language: the language of the target tree selector: the selector of the target tree

process_ttree(*ttree*)
Mark all subjects in a sentence

alex.components.nlg.tectotpl.block.t2a.cs.markverbalcategories module

class alex.components.nlg.tectotpl.block.t2a.cs.markverbalcategories.**MarkVerbalCategories**(*scer*
args

Bases: *alex.components.nlg.tectotpl.core.block.Block*

Finishes marking synthetic verbal categories: tense, finiteness, mood.

Arguments: language: the language of the target tree selector: the selector of the target tree

mark_subpos_tense(*tnode*, *anode*)
Marks the Sub-POS and tense parts of the morphcat structure in plain verbal a-nodes.

process_tnode(*tnode*)
Marks verbal categories for a t-node.

resolve_imperative(*anode*)
Mark an imperative a-node.

resolve_infinitive(*anode*)
Mark an infinitive a-node correctly.

alex.components.nlg.tectotpl.block.t2a.cs.movecliticstowackernagel module

class alex.components.nlg.tectotpl.block.t2a.cs.movecliticstowackernagel.**MoveCliticsToWackerr**

Bases: *alex.components.nlg.tectotpl.core.block.Block*

Move clitics (e.g. 'se', 'to' etc.) to the second (Wackernagel) position in the clause.

clitic_order(*clitic*)
Return the position of the given clitic in the natural Czech order of multiple clitics in the same clause.

find_eo1st_pos(*clause_root*, *clause_1st*)
Find the last word before the Wackernagel position.

handle_pronoun_je(*anode*)
If the given node is a personal pronoun with the form 'je', move it before its parent's subtree and return True. Return false otherwise.

is_clitic(*anode*)
Return True if the given node belongs to a clitic.

is_coord_taking_1st_pos(*clause_root*)
Return True if the clause root is a coordination member and the coordinating conjunction or shared subjunction is taking up the 1st position. E.g. 'Běžel, aby se zahřál a dostal se dřív domů.'

process_atree(*aroot*)
Process the individual clauses – find and move clitics within them.

process_clause(*clause*)
Find and move clitics within one clause.

should_ignore(*anode*, *clause_number*)
Return True if this word should be ignored in establishing the Wackernagel position.

verb_group_root(*clitic*)
Find the root of the verbal group that the given clitic belongs to. If the verbal group is governed by a conjunction, return this conjunction.

alex.components.nlg.tectotpl.block.t2a.cs.projectclausenumber module

class alex.components.nlg.tectotpl.block.t2a.cs.projectclausenumber.**ProjectClauseNumber**(*scenar*
args)

Bases: *alex.components.nlg.tectotpl.core.block.Block*

Project clause numbering from t-nodes to a-nodes.

Arguments: language: the language of the target tree selector: the selector of the target tree

process_tnode(*tnode*)
Project the t-node's clause number to all its corresponding a-nodes.

alex.components.nlg.tectotpl.block.t2a.cs.reversenumbernoundependency module

class alex.components.nlg.tectotpl.block.t2a.cs.reversenumbernoundependency.**ReverseNumberNour**

Bases: *alex.components.nlg.tectotpl.core.block.Block*

This block reverses the dependency of incongruent Czech numerals (5 and higher), hanging their parents under them in the a-tree.

Arguments: language: the language of the target tree selector: the selector of the target tree

process_ttree(*ttree*)
Rehang the numerals for the given t-tree & a-tree pair

alex.components.nlg.tectotpl.block.t2a.cs.vocalizeprepos module

class alex.components.nlg.tectotpl.block.t2a.cs.vocalizeprepos.**VocalizePrepos**(*scenario*,
args)

Bases: *alex.components.nlg.tectotpl.core.block.Block*

This block replaces the forms of prepositions 'k', 'v', 'z', 's' with their vocalized variants 'ke'/'ku', 've', 'ze', 'se' according to the following word.

process_atree(*aroot*)
Find and vocalize prepositions according to their context.

vocalize(*prep*, *follow*)
Given a preposition lemma and the form of the word following it, return the appropriate form (base or vocalized).

Module contents

Submodules

alex.components.nlg.tectotpl.block.t2a.addauxwords module

class alex.components.nlg.tectotpl.block.t2a.addauxwords.**AddAuxWords**(*scenario*,
args)

Bases: *alex.components.nlg.tectotpl.core.block.Block*

Add auxiliary a-nodes according to formemes.

This is a base class for all steps adding auxiliary nodes.

Arguments: language: the language of the target tree selector: the selector of the target tree

get_anode(*tnode*)
Return the a-node corresponding to the given t-node. Defaults to lexical a-node.

get_aux_forms(*tnode*)
This should return a list of new forms for the auxiliaries, or None if none should be added

new_aux_node(*aparent*, *form*)
Create an auxiliary node with the given surface form and parent.

postprocess(*tnode*, *anode*, *aux_nodes*)
Apply content-specific post-processing to the newly created auxiliary a-nodes (to be overridden if needed).

process_tnode(*tnode*)
Add auxiliary words to the a-layer for a t-node.

alex.components.nlg.tectotpl.block.t2a.copyttree module

class alex.components.nlg.tectotpl.block.t2a.copyttree.**CopyTTree**(*scenario*, *args*)
Bases: *alex.components.nlg.tectotpl.core.block.Block*

This block creates an a-tree based on a t-tree in the same zone.

Arguments: language: the language of the target zone selector: the selector of the target zone

copy_subtree(*troot*, *aroot*)
Deep-copy a subtree, creating nodes with the same attributes, but different IDs.

process_zone(*zone*)
Starting tree copy

alex.components.nlg.tectotpl.block.t2a.imposeagreement module

class alex.components.nlg.tectotpl.block.t2a.imposeagreement.**ImposeAgreement**(*scenario*, *args*)
Bases: *alex.components.nlg.tectotpl.core.block.Block*

A common ancestor for blocks that impose a grammatical agreement of some kind: they should override the should_agree(tnode), process_excepts(tnode), and impose(tnode) methods.

Arguments: language: the language of the target tree selector: the selector of the target tree

impose(*tnode*, *match_nodes*)
Impose the agreement onto the given (regular) node.

process_excepts(*tnode*, *match_nodes*)
Process exceptions from the agreement. If an exception has been found and impose() should not fire, return True.

process_tnode(*tnode*)
Impose the required agreement on a node, if applicable.

should_agree(*tnode*)
Check whether the agreement applies to the given node; if so, return the relevant nodes this node should agree with.

Module contents

alex.components.nlg.tectotpl.block.t2t package

Module contents

alex.components.nlg.tectotpl.block.util package

Submodules

alex.components.nlg.tectotpl.block.util.copytree module

class alex.components.nlg.tectotpl.block.util.copytree.**CopyTree**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.core.block.Block*

This block is able to copy a tree on the same layer from a different zone.

Arguments: language: the language of the TARGET zone selector: the selector of the TARGET zone source_language the language of the SOURCE zone (defaults to same as target) source_selector the selector of the SOURCE zone (defaults to same as target) layer: the layer to which this conversion should be applied

TODO: apply to more layers at once

copy_subtree(*source_root*, *target_root*)

Deep-copy a subtree, creating nodes with the same attributes, but different IDs

process_bundle(*bundle*)

For each bundle, copy the tree on the given layer in the given zone to another zone.

alex.components.nlg.tectotpl.block.util.eval module

class alex.components.nlg.tectotpl.block.util.eval.**Eval**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.core.block.Block*

This block executes arbitrary Python code for each document/bundle or each zone/tree/node matching the current language and selector.

Arguments:

document, bundle, zone, atree, anode, ttree, tnode, ntree, nnode, ptree, pnode: code to execute for each <name of the argument>

Arguments may be combined, but at least one of them must be set. If only X<tree/node> are set, language and selector is required.

process_bundle(*bundle*)

Process a document (execute code from the 'bundle' argument and dive deeper)

process_document(*doc*)

Process a document (execute code from the 'document' argument and dive deeper)

process_zone(*zone*)

Process a zone (according to language and selector; execute code for the zone or X<tree|node>) arguments)

valid_args = [u'document', u'doc', u'bundle', u'zone', u'atree', u'anode', u'ttree', u'tnode', u'ntree', u'nnode', u'ptre

alex.components.nlg.tectotpl.block.util.setglobal module

class alex.components.nlg.tectotpl.block.util.setglobal.**SetGlobal**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.core.block.Block*

process_bundle(*doc*)

This block does nothing with the documents, its only work is setting the global arguments in the initialization phase.

Module contents

alex.components.nlg.tectotpl.block.write package

Submodules

alex.components.nlg.tectotpl.block.write.basewriter module

class alex.components.nlg.tectotpl.block.write.basewriter.**BaseWriter**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.core.block.Block*

Base block for output writing.

get_output_file_name(*doc*)

Create an output file name for the given document.

alex.components.nlg.tectotpl.block.write.yaml module

class alex.components.nlg.tectotpl.block.write.yaml.**YAML**(*scenario*, *args*)

Bases: *alex.components.nlg.tectotpl.block.write.basewriter.BaseWriter*

default_extension = u'.yaml'

process_document(*doc*)

Write a YAML document

serialize_bundle(*bundle*)

Serialize a bundle to a list.

serialize_node(*node*, *add_parent_id*)

Serialize a node to a hash; using the correct attributes for the tree type given. Add the node parent's id if needed.

serialize_tree(*root*)

serialize_zone(*zone*)

Serialize a zone into a hash

Module contents

Module contents

alex.components.nlg.tectotpl.core package

Submodules

alex.components.nlg.tectotpl.core.block module

class alex.components.nlg.tectotpl.core.block.**Block**(*scenario*, *args*)

Bases: object

A common ancestor to all Treex processing blocks.

load()
Load required files / models, to be overridden by child blocks.

process_bundle(*bundle*)
Process a bundle. Default behavior is to process the zone according to the current language and selector.

process_document(*doc*)
Process a document. Default behavior is to look for methods that process a bundle/zone/tree/node. If none is found, raise a NotImplementedError.

process_zone(*zone*)
Process a zone. Default behavior is to try if there is a process_Xtree or process_Xnode method and run this method, otherwise raise an error.

alex.components.nlg.tectotpl.core.document module

class alex.components.nlg.tectotpl.core.document.**Bundle**(*document*, *data=None*, *b_ord=None*)

Bases: object

Represents a bundle, i.e. a list of zones pertaining to the same sentence (in different variations).

create_zone(*language*, *selector*)
Creates a zone at the given language and selector. Will overwrite any existing zones.

document
The document this bundle belongs to.

get_all_zones()
Return all zones contained in this bundle.

get_or_create_zone(*language*, *selector*)
Returns the zone for a language and selector; if it does not exist, creates an empty zone.

get_zone(*language*, *selector*)
Returns the corresponding zone for a language and selector; raises an exception if the zone does not exist.

has_zone(*language*, *selector*)
Returns True if the bundle has a zone for the given language and selector.

ord
The order of this bundle in the document, as given by constructor

class alex.components.nlg.tectotpl.core.document.**Document**(*filename=None*, *data=None*)

Bases: object

This represents a Treex document, i.e. a sequence of bundles. It contains an index of node IDs.

create_bundle(*data=None*)
Append a new bundle and return it.

get_node_by_id(*node_id*)

index_backref(*attr_name*, *source_id*, *target_ids*)
Keep track of a backward reference (source, target node IDs are in the direction of the original reference)

index_node(*node*)
Index a node by its id. Also index the node's references in the backwards reference index.

remove_backref(*attr_name*, *source_id*, *target_ids*)
Remove references from the backwards index.

remove_node(*node_id*)
Remove a node from all indexes.

class alex.components.nlg.tectotpl.core.document.**Zone**(*data=None*, *language=None*, *selector=None*, *bundle=None*)
Bases: object

Represents a zone, i.e. a sentence and corresponding trees.

atree
Direct access to a-tree (will raise an exception if the tree does not exist).

bundle
The bundle in which this zone is located

create_atree()
Create a tree on the a-layer

create_ntree()
Create a tree on the n-layer

create_ptree()
Create a tree on the p-layer

create_tree(*layer*, *data=None*)
Create a tree on the given layer, filling it with the given data (if applicable).

create_ttree()
Create a tree on the t-layer

document
The document in which this zone is located

get_tree(*layer*)
Return a tree this node has on the given layer or raise an exception if the tree does not exist.

has_atree()
Return true if this zone has an a-tree.

has_ntree()
Return true if this zone has an n-tree.

has_ptree()
Return true if this zone has a p-tree.

has_tree(*layer*)
Return True if this zone has a tree on the given layer, False otherwise.

has_ttree()
Return true if this zone has a t-tree.

language_and_selector
Return string concatenation of the zone's language and selector.

ntree
Direct access to n-tree (will raise an exception if the tree does not exist).

ptree
Direct access to p-tree (will raise an exception if the tree does not exist).

ttree
Direct access to t-tree (will raise an exception if the tree does not exist).

alex.components.nlg.tectotpl.core.exception module

exception alex.components.nlg.tectotpl.core.exception.**DataException**(*path*)
Bases: *alex.components.nlg.tectotpl.core.exception.TreexException*

Data file not found exception

exception alex.components.nlg.tectotpl.core.exception.**LoadingException**(*text*)
Bases: *alex.components.nlg.tectotpl.core.exception.TreexException*

Block loading exception

exception alex.components.nlg.tectotpl.core.exception.**RuntimeException**(*text*)
Bases: *alex.components.nlg.tectotpl.core.exception.TreexException*

Block runtime exception

exception alex.components.nlg.tectotpl.core.exception.**ScenarioException**(*text*)
Bases: *alex.components.nlg.tectotpl.core.exception.TreexException*

Scenario-related exception.

exception alex.components.nlg.tectotpl.core.exception.**TreexException**(*message*)
Bases: exceptions.Exception

Common ancestor for Treex exception

alex.components.nlg.tectotpl.core.log module

alex.components.nlg.tectotpl.core.log.**log_info**(*message*)
Print an information message

alex.components.nlg.tectotpl.core.log.**log_warn**(*message*)
Print a warning message

alex.components.nlg.tectotpl.core.node module

class alex.components.nlg.tectotpl.core.node.**A**(*data=None*, *parent=None*, *zone=None*)
Bases: *alex.components.nlg.tectotpl.core.node.Node*, *alex.components.nlg.tectotpl.core.node*
alex.components.nlg.tectotpl.core.node.EffectiveRelations,
alex.components.nlg.tectotpl.core.node.InClause

Representing an a-node

attrib = [(u'form', <type 'unicode'>), (u'lemma', <type 'unicode'>), (u'tag', <type 'unicode'>), (u'afun', <type 'unicod

is_coap_root()

morphcat_case

morphcat_gender

morphcat_grade

morphcat_members = [u'pos', u'subpos', u'gender', u'number', u'case', u'person', u'tense', u'negation', u'voice', u'gr

morphcat_mood

morphcat_negation

morphcat_number

morphcat_person

morphcat_pos

morphcat_possgender

morphcat_possnumber

morphcat_subpos

morphcat_tense

morphcat_voice

ref_attrib = [u'p_terminal.rf']

reset_morphcat()
Reset the morphcat structure members to '.'

class alex.components.nlg.tectotpl.core.node.**EffectiveRelations**
Bases: object

Representing a node with effective relations

attrib = [(u'is_member', <type 'bool'>)]

get_coap_members()
Return the members of the coordination, if the node is a coap root. Otherwise return the node itself.

get_echildren(*or_topological=False*, *add_self=False*, *ordered=False*, *preceding_only=False*, *following_only=False*)
Return the effective children of the current node.

get_eparents(*or_topological=False*, *add_self=False*, *ordered=False*, *preceding_only=False*, *following_only=False*)
Return the effective parents of the current node.

is_coap_root()
Testing whether the node is a coordination/apposition root. Must be implemented in descendants.

ref_attrib = []

class alex.components.nlg.tectotpl.core.node.**InClause**
Bases: object

Represents nodes that are organized in clauses

attrib = [(u'clause_number', <type 'int'>), (u'is_clause_head', <type 'bool'>)]

get_clause_root()
Return the root of the clause the current node resides in.

ref_attrib = []

class alex.components.nlg.tectotpl.core.node.**N**(*data=None*, *parent=None*, *zone=None*)
Bases: *alex.components.nlg.tectotpl.core.node.Node*

Representing an n-node

attrib = [(u'ne_type', <type 'unicode'>), (u'normalized_name', <type 'unicode'>), (u'a.rf', <type 'list'>)]

ref_attrib = [u'a.rf']

class alex.components.nlg.tectotpl.core.node.**Node**(*data=None*, *parent=None*, *zone=None*)
Bases: object

Representing a node in a tree (recursively)

attrib = [(u'alignment', <type 'list'>), (u'wild', <type 'dict'>)]

create_child(*id=None*, *data=None*)
Create a child of the current node

document
The document this node is a member of.

get_attr(*name*)
Return the value of the given attribute. Allows for dictionary nesting, e.g. 'morphcat/gender'

get_attr_list(*include_types=False*, *safe=False*)
Get attributes of the current class (gathering all attributes of base classes)

get_children(*add_self=False*, *ordered=False*, *preceding_only=False*, *following_only=False*)
Return all children of the node

get_depth()
Return the depth, i.e. the distance to the root.

get_deref_attr(*name*)
This assumes the given attribute holds node id(s) and returns the corresponding node(s)

get_descendants(*add_self=False*, *ordered=False*, *preceding_only=False*, *following_only=False*)
Return all topological descendants of this node.

get_ref_attr_list(*split_nested=False*)
Return a list of the attributes of the current class that contain references (splitting nested ones, if needed)

get_referenced_ids()
Return all ids referenced by this node, keyed under their reference types in a hash.

id
The unique id of the node within the document.

is_root
Return true if this node is a root

parent
The parent of the current node. None for roots.

ref_attrib = []

remove()
Remove the node from the tree.

remove_reference(*ref_type*, *refd_id*)
Remove the reference of the given type to the given node.

root
The root of the tree this node is in.

set_attr(*name*, *value*)
Set the value of the given attribute. Allows for dictionary nesting, e.g. 'morphcat/gender'

set_deref_attr(*name*, *value*)
This assumes the value is a node/list of nodes and sets its id/their ids as the value of the given attribute.

zone
The zone this node belongs to.

class alex.components.nlg.tectotpl.core.node.**Ordered**
Bases: object

Representing an ordered node (has an attribute called ord), defines sorting.

attrib = [(u'ord', <type 'int'>)]

get_next_node()
Get the following node in the ordering.

get_prev_node()
Get the preceding node in the ordering.

is_first_node()
Return True if this node is the first node in the tree, i.e. has no previous nodes.

is_last_node()
Return True if this node is the last node in the tree, i.e. has no following nodes.

is_right_child
Return True if this node has a greater ord than its parent. Returns None for a root.

ref_attrib = []

shift_after_node(*other*, *without_children=False*)
Shift one node after another in the ordering.

shift_after_subtree(*other*, *without_children=False*)
Shift one node after the whole subtree of another node in the ordering.

shift_before_node(*other*, *without_children=False*)
Shift one node before another in the ordering.

shift_before_subtree(*other*, *without_children=False*)
Shift one node before the whole subtree of another node in the ordering.

class alex.components.nlg.tectotpl.core.node.**P**(*data=None*, *parent=None*, *zone=None*)
Bases: *alex.components.nlg.tectotpl.core.node.Node*

Representing a p-node

attrib = [(u'is_head', <type 'bool'>), (u'index', <type 'unicode'>), (u'coindex', <type 'unicode'>), (u'edgelabel', <type '

ref_attrib = []

class alex.components.nlg.tectotpl.core.node.**T**(*data=None*, *parent=None*, *zone=None*)
Bases: *alex.components.nlg.tectotpl.core.node.Node*, *alex.components.nlg.tectotpl.core.node*
alex.components.nlg.tectotpl.core.node.EffectiveRelations,
alex.components.nlg.tectotpl.core.node.InClause

Representing a t-node

add_aux_anodes(*new_anodes*)
Add an auxiliary a-node/a-nodes to the list.

anodes
Return all anodes of a t-node

attrib = [(u'functor', <type 'unicode'>), (u'formeme', <type 'unicode'>), (u't_lemma', <type 'unicode'>), (u'nodetype',

aux_anodes

compl_nodes

coref_gram_nodes

coref_text_nodes

gram_aspect

gram_degcmp

gram_deontmod

gram_diathesis

gram_dispmod

gram_gender

gram_indeftype

gram_iterativeness

gram_negation

gram_number

gram_numertype

gram_person

gram_politeness

gram_resultative

gram_sempos

gram_tense

gram_verbmod

is_coap_root()

lex_anode

ref_attrib = [u'a/lex.rf', u'a/aux.rf', u'compl.rf', u'coref_gram.rf', u'coref_text.rf']

remove_aux_anodes(*to_remove*)
Remove an auxiliary a-node from the list

alex.components.nlg.tectotpl.core.run module

class alex.components.nlg.tectotpl.core.run.**Scenario**(*config*)
Bases: object

This represents a scenario, i.e. a sequence of blocks to be run on the data

apply_to(*string*, *language=None*, *selector=None*)
Apply the whole scenario to a string (which should be readable by the first block of the scenario), return the sentence(s) of the given target language and selector.

load_blocks()
Load all blocks into memory, finding and creating class objects.

alex.components.nlg.tectotpl.core.util module

alex.components.nlg.tectotpl.core.util.**as_list**(*value*)
Cast anything to a list (just copy a list or a tuple, or put an atomic item to as a single element to a list).

alex.components.nlg.tectotpl.core.util.**file_stream**(*filename*, *mode=u'r'*, *encoding=u'UTF-8'*)
Given a file stream or a file name, return the corresponding stream, handling GZip. Depending on mode, open an input or output stream.

alex.components.nlg.tectotpl.core.util.**first**(*condition_function*, *sequence*, *default=None*)
Return first item in sequence where condition_function(item) == True, or None if no such item exists.

Module contents

alex.components.nlg.tectotpl.tool package

Subpackages

alex.components.nlg.tectotpl.tool.lexicon package

Submodules

alex.components.nlg.tectotpl.tool.lexicon.cs module

class alex.components.nlg.tectotpl.tool.lexicon.cs.**Lexicon**
Bases: object

get_possessive_adj_for(*noun_lemma*)
Given a noun lemma, this returns a possessive adjective if it's in the database.

has_expletive(*lemma*)
Return an expletive for a 'že'-clause that this verb governs, or False. Lemmas must include reflexive particles for reflexiva tantum.

has_synthetic_future(*verb_lemma*)
Returns True if the verb builds a synthetic future tense form with the prefix 'po-'/'pů-'.

inflect_conditional(*lemma*, *number*, *person*)
Return inflected form of a conditional particle/conjunction

is_coord_conj(*lemma*)
Return 'Y'/'N' if the given lemma is a coordinating conjunction (depending on whether one should write a comma directly in front).

is_incongruent_numeral(*numeral*)
Return True if the given lemma belongs to a Czech numeral that takes a genitive attribute instead of being an attribute itself

is_named_entity_label(*lemma*)
Return 'I'/'C' if the given lemma is a named entity label (used as congruent/incongruent attribute).

is_personal_role(*lemma*)
Return true if the given lemma is a personal role.

load_possessive_adj_dict(*data_dir*)
Read the possessive-adjective-to-noun conversion file and save it to the database.

number_for(*numeral*)
Given a Czech numeral, returns the corresponding number.

Module contents

alex.components.nlg.tectotpl.tool.ml package

Submodules

alex.components.nlg.tectotpl.tool.ml.dataset module Data set representation with ARFF input possibility.

class alex.components.nlg.tectotpl.tool.ml.dataset.**Attribute**(*name*, *type_spec*)
Bases: object

This represents an attribute of the data set.

get_arff_type()
Return the ARFF type of the given attribute (numeric, string or list of values for nominal attributes).

num_values
Return the number of distinct values found in this attribute. Returns -1 for numeric attributes where the number of values is not known.

numeric_value(*value*)
Return a numeric representation of the given value. Raise a ValueError if the given value does not conform to the attribute type.

soft_numeric_value(*value*, *add_values*)
Same as numeric_value(), but will not raise exceptions for unknown numeric/string values. Will either add the value to the list or return a NaN (depending on the add_values setting).

value(*numeric_val*)
Given a numeric (int/float) value, returns the corresponding string value for string or nominal attributes, or the identical value for numeric attributes. Returns None for missing nominal/string values, NaN for missing numeric values.

values_set()
Return a set of all possible values for this attribute.

class alex.components.nlg.tectotpl.tool.ml.dataset.**DataSet**
Bases: object

ARFF relation data representation.

DENSE_FIELD = u'([^"\\\']|[^,]*|\\\'[^\\\']*(\\\\\\\'[^\\\']*)*(?<!\\\\)\\\'|"[^"]*(\\\\"[^"]*)*(?<!\\\\)"),'

SPARSE_FIELD = u'([0-9]+)\\s+([^"\\\'\\s][^,]*|\\\'[^\\\']*(\\\\\\\'[^\\\']*)*\\\'|"[^"]*(\\\\"[^"]*)*"),'

SPEC_CHARS = u'[\\n\\r\\\'"\\\\\\t%]'

add_attrib(*attrib*, *values=None*)
Add a new attribute to the data set, with pre-filled values (or missing, if not set).

append(*other*)
Append instances from one data set to another. Their attributes must be compatible (of the same types).

as_bunch(*target*, *mask_attrib=[]*, *select_attrib=[]*)
Return the data as a scikit-learn Bunch object. The target parameter specifies the class attribute.

as_dict(*mask_attrib=[]*, *select_attrib=[]*)
Return the data as a list of dictionaries, which is useful as an input to DictVectorizer.

Attributes (numbers or indexes) listed in mask_attrib are not added to the dictionary. Missing values are also not added to the dictionary. If mask_attrib is not set but select_attrib is set, only attributes listed in select_attrib are added to the dictionary.

attrib_as_vect(*attrib*, *dtype=None*)
Return the specified attribute (by index or name) as a list of values. If the data type parameter is left as default, the type of the returned values depends on the attribute type (strings for nominal or string attributes, floats for numeric ones). Set the data type parameter to int or float to override the data type.

attrib_index(*attrib_name*)
Given an attribute name, return its number. Given a number, return precisely that number. Return -1 on failure.

delete_attrib(*attribs*)
Given a list of attributes, delete them from the data set. Accepts a list of names or indexes, or one name, or one index.

filter(*filter_func*, *keep_copy=True*)
Filter the data set using a filtering function and return a filtered data set.

The filtering function must take two arguments - current instance index and the instance itself in an attribute-value dictionary form - and return a boolean.

If keep_copy is set to False, filtered instances will be removed from the original data set.

get_attrib(*attrib*)
Given an attribute name or index, return the Attribute object.

get_headers()
Return a copy of the headers of this data set (just attributes list, relation name and sparse/dense setting)

instance(*index*, *dtype=u'dict'*, *do_copy=True*)
Return the given instance as a dictionary (or a list, if specified).

If do_copy is set to False, do not create a copy of the list for dense instances (other types must be copied anyway).

is_empty
Return true if the data structures are empty.

load_from_arff(*filename*, *encoding=u'UTF-8'*)
Load an ARFF file/stream, filling the data structures.

load_from_dict(*data*, *attrib_types={}*)
Fill in values from a list of dictionaries (=instances). Attributes are assumed to be of string type unless specified otherwise in the attrib_types variable. Currently only capable of creating dense data sets.

load_from_matrix(*attr_list*, *matrix*)
Fill in values from a matrix.

load_from_vect(*attrib*, *vect*)
Fill in values from a vector of values and an attribute (allow adding values for nominal attributes).

match_headers(*other*, *add_values=False*)
Force this data set to have equal headers as the other data set. This cares for different values of nominal/numeric attributes – (numeric values will be the same, values unknown in the other data set will be set to NaNs). In other cases, such as a different number or type of attributes, an exception is thrown.

merge(*other*)
Merge two DataSet objects. The list of attributes will be concatenated. The two data sets must have the same number of instances and be either both sparse or both non-sparse.

Instance weights are left unchanged (from this data set).

rename_attrib(*old_name*, *new_name*)
Rename an attribute of this data set (find it by original name or by index).

save_to_arff(*filename*, *encoding=u'UTF-8'*)
Save the data set to an ARFF file

separate_attrib(*attribs*)
Given a list of attributes, delete them from the data set and return them as a new separate data set. Accepts a list of names or indexes, or one name, or one index.

split(*split_func*, *keep_copy=True*)
Split the data set using a splitting function and return a dictionary where keys are different return values of the splitting function and values are data sets containing instances which yield the respective splitting function return values.

The splitting function takes two arguments - the current instance index and the instance itself as an attribute-value dictionary. Its return value determines the split.

If keep_copy is set to False, ALL instances will be removed from the original data set.

subset (**args*, ***kwargs*)
Return a data set representing a subset of this data set's values.

Args can be a slice or [start,] stop [, stride] to create a slice. No arguments result in a complete copy of the original.

Kwargs may contain just one value – if copy is set to false, the sliced values are removed from the original data set.

value (*instance*, *attr_idx*)
Return the value of the given instance and attribute.

class alex.components.nlg.tectotpl.tool.ml.dataset.**DataSetIterator** (*dataset*)
Bases: object

An iterator over the instances of a data set.

next ()
Move to the next instance.

alex.components.nlg.tectotpl.tool.ml.model module

class alex.components.nlg.tectotpl.tool.ml.model.**AbstractModel** (*config*)
Bases: object

Abstract ancestor of different model classes

check_classification_input (*instances*)
Check classification input data format, convert to list if needed.

classify (*instances*)
This must be implemented in derived classes.

evaluate (*test_file*, *encoding=u'UTF-8'*, *classif_file=None*)
Evaluate on the given test data file. Return accuracy. If classif_file is set, save the classification results to this file.

get_classes (*data*, *dtype=<type 'int'>*)
Return a vector of class values from the given DataSet. If dtype is int, the integer values are returned. If dtype is None, the string values are returned.

static load_from_file (*model_file*)
Load the model from a pickle file or stream (supports GZip compression).

load_training_set (*filename*, *encoding=u'UTF-8'*)
Load the given training data set into memory and strip it if configured to via the train_part parameter.

save_to_file (*model_file*)
Save the model to a pickle file or stream (supports GZip compression).

class alex.components.nlg.tectotpl.tool.ml.model.**Model** (*config*)
Bases: *alex.components.nlg.tectotpl.tool.ml.model.AbstractModel*

PREDICTED = u'PREDICTED'

classify (*instances*)
Classify a set of instances (possibly one member).

construct_classifier(*cfg*)
Given the config file, construct the classifier (based on the 'classifier' or 'classifier_class'/'classifier_params' settings. Defaults to DummyClassifier.

static create_training_job(*config*, *work_dir*, *train_file*, *name=None*, *memory=8*, *encoding=u'UTF-8'*)
Submit a training process on the cluster which will save the model to a pickle. Return the submitted job and the future location of the model pickle. train_file cannot be a stream, it must be an actual file.

train(*train_file*, *encoding=u'UTF-8'*)
Train the model on the specified training data file.

train_on_data(*train*)
Train model on the specified training data set (which must be a loaded DataSet object).

class alex.components.nlg.tectotpl.tool.ml.model.**SplitModel**(*config*)
Bases: *alex.components.nlg.tectotpl.tool.ml.model.AbstractModel*

A model that's actually composed of several Model-s.

classify(*instances*)
Classify a set of instances.

train(*train_file*, *work_dir*, *memory=8*, *encoding=u'UTF-8'*)
Read training data, split them and train the individual models (in cluster jobs).

Module contents

Submodules

alex.components.nlg.tectotpl.tool.cluster module

class alex.components.nlg.tectotpl.tool.cluster.**Job**(*code=None*, *header=u'#!/usr/bin/env pythonn# coding=utf8nfrom __future__ import unicode_literalsn'*, *name=None*, *work_dir=None*, *dependencies=None*)
Bases: object

This represents a piece of code as a job on the cluster, holds information about the job and is able to retrieve job metadata.

The most important method is submit(), which submits the given piece of code to the cluster.

Important attributes (some may be set in the constructor or at job submission, but all may be set between construction and launch): ———————————————————— name – job name on the cluster (and the name of the created

Python script, default will be generated if not set)

code – the Python code to be run (needs to have imports and sys.path set properly)

header – the header of the created Python script (may contain imports etc.)

memory – the amount of memory to reserve for this job on the cluster

cores – the number of cores needed for this job work_dir – the working directory where the job script will be

created and run (will be created on launch)

dependencies-list of Jobs this job depends on (must be submitted before submitting this job)

In addition, the following values may be queried for each job at runtime or later: ———————————————————— submitted – True if the job has been submitted to the cluster. state – current job state ('qw' = queued, 'r' = running, 'f'

= finished, only if the job was submitted)

host – the machine where the job is running (short name) jobid – the numeric id of the job in the cluster (NB: type is

string!)

report – job report using the qacct command (dictionary, available only after the job has finished)

exit_status- numeric job exit status (if the job is finished)

DEFAULT_CORES = 1

DEFAULT_HEADER = u'#!/usr/bin/env python\n# coding=utf8\nfrom __future__ import unicode_literals\n'

DEFAULT_MEMORY = 4

DIR_PREFIX = u'_clrun-'

FINISH = u'f'

JOBNAME_LEGAL_CHARS = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'

NAME_PREFIX = u'pyjob_'

QSUB_MEMORY_CMD = u'-hard -l mem_free={0} -l act_mem_free={0} -l h_vmem={0}'

QSUB_MULTICORE_CMD = u'-pe smp {0}'

TIME_POLL_DELAY = 60

TIME_QUERY_DELAY = 1

add_dependency(*dependency*)
Adds a dependency on the given Job(s).

exit_status
Retrieve the exit status of the job via the qacct report. Throws an exception the job is still running and the exit status is not known.

get_script_text()
Join headers and code to create a meaningful Python script.

host
Retrieve information about the host this job is/was running on.

jobid
Return the job id.

name
Return the job name.

remove_dependency(*dependency*)
Removes the given Job(s) from the dependencies list.

report
Access to qacct report. Please note that running the qacct command takes a few seconds, so the first access to the report is rather slow.

state
Retrieve information about current job state. Will also retrieve the host this job is running on and store it in the __host variable, if applicable.

submit (*memory=None*, *cores=None*, *work_dir=None*)
Submit the job to the cluster. Override the pre-set memory and cores defaults if necessary. The job code, header and working directory must be set in advance. All jobs on which this job is dependent must already be submitted!

wait (*poll_delay=None*)
Waits for the job to finish. Will raise an exception if the job did not finish successfully. The poll_delay variable controls how often the job state is checked.

Module contents

Module contents

alex.components.nlg.tools package

Submodules

alex.components.nlg.tools.cs module A collection of helper functions for generating Czech.

class alex.components.nlg.tools.cs.**CzechTemplateNLGPostprocessing**
Bases: *alex.components.nlg.template.TemplateNLGPostprocessing*

Postprocessing filled in NLG templates for Czech.

Currently, this class only handles preposition vocalization.

postprocess (*nlg_text*)

vocalize_prepos (*text*)
Vocalize prepositions in the utterance, i.e. 'k', 'v', 'z', 's' are changed to 'ke', 've', 'ze', 'se' if appropriate given the following word.

This is mainly needed for time expressions, e.g. "v jednu hodinu" (at 1:00), but "ve dvě hodiny" (at 2:00).

alex.components.nlg.tools.cs.**vocalize_prep** (*prep*, *following_word*)
Given a base for of a preposition and the form of the word following it, return the appropriate form (base or vocalized).

Case insensitive; however, the returned vocalization is always lowercase.

alex.components.nlg.tools.cs.**word_for_number** (*number*, *categ=u'M1'*)
Returns a word given a number 1-100 (in the given gender + case). Gender (M, I, F, N) and case (1-7) are given concatenated.

alex.components.nlg.tools.en module A collection of helper functions for generating English.

alex.components.nlg.tools.en.**every_word_for_number** (*number*, *ordinary=False*, *use_coupling=False*)

params: ordinary - if set to True, it returns ordinal of the number (fifth rather than five etc).

use_coupling if set to True, it returns number greater than 100 with "and" between hundreds and tens (two hundred and seventeen rather than two hundred seventeen).

Returns a word given a number 1-100

alex.components.nlg.tools.en.**word_for_number**(*number*, *ordinary=False*)
Returns a word given a number 1-100

Module contents

Submodules

alex.components.nlg.autopath module self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing "pypy" is in sys.path.

If you modify the master "autopath.py" version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

alex.components.nlg.common module
alex.components.nlg.common.**get_nlg_type**(*cfg*)
alex.components.nlg.common.**nlg_factory**(*nlg_type*, *cfg*)

alex.components.nlg.exceptions module
exception alex.components.nlg.exceptions.**NLGException**
Bases: *alex.AlexException*
exception alex.components.nlg.exceptions.**TemplateNLGException**
Bases: *alex.components.nlg.exceptions.NLGException*

alex.components.nlg.template module
class alex.components.nlg.template.**AbstractTemplateNLG**(*cfg*)
Bases: object

Base abstract class for template-filling generators, providing the routines for template loading and selection.

The generation (i.e. template filling) is left to the derived classes.

It implements numerous backoff strategies: 1) it matches the exactly the input dialogue against the templates 2) if it cannot find exact match, then it tries to find a generic template (slot-independent) 3) if it cannot find a generic template, the it tries to compose

the template from templates for individual dialogue act items

backoff(*da*)
Provide an alternative NLG template for the dialogue output which is not covered in the templates. This serves as a backoff solution. This should be implemented in derived classes.

compose_utterance_greedy(*da*)
Compose an utterance from templates by iteratively looking for the longest (up to self.compose_greedy_lookahead) matching sub-utterance at the current position in the DA.

Returns the composed utterance.

compose_utterance_single(*da*)
Compose an utterance from templates for single dialogue act items. Returns the composed utterance.

fill_in_template(*tpl*, *svs*)
Fill in the given slot values of a dialogue act into the given template. This should be implemented in derived classes.

generate(*da*)
Generate the natural text output for the given dialogue act.

First, try to find an exact match with no variables to fill in. Then try to find a relaxed match of a more generic template and fill in the actual values of the variables.

get_generic_da(*da*)
Given a dialogue act and a list of slots and values, substitute the generic values (starting with { and ending with }) with empty string.

get_generic_da_given_svs(*da*, *svs*)
Given a dialogue act and a list of slots and values, substitute the matching slot and values with empty string.

load_templates(*file_name*)
Load templates from an external file, which is assumed to be a Python source which defines the variable 'templates' as a dictionary containing stringified dialog acts as keys and (lists of) templates as values.

match_and_fill_generic(*da*, *svs*)
Match a generic template and fill in the proper values for the slots which were substituted by a generic value.

Will return the output text with the proper values filled in if a generic template can be found; will throw a TemplateNLGException otherwise.

match_generic_templates(*da*, *svs*)
Find a matching template for a dialogue act using substitutions for slot values.

Returns a matching template and a dialogue act where values of some of the slots are substituted with a generic value.

random_select(*tpl*)
Randomly select alternative templates for generation.

The selection process is modeled by an embedded list structure (a tree-like structure). In the first level, the algorithm selects one of N. In the second level, for every item it selects one of M, and joins them together. This continues toward the leaves which must be non-list objects.

There are the following random selection options (only the first three):

1.{ 'hello()' : u"Hello", }

   This will return the "Hello" string.

2.{ 'hello()' : (u"Hello",

        u"Hi",

    ),

   }

   This will return one of the "Hello" or "Hi" strings.

2.{ 'hello()' : (

```
        [
          (u"Hello.", u"Hi.",
          ) (u"How are you doing?",
             u"Welcome".,
          ), u"Speak!",
        ],
        u"Hi my friend."
      ),
}
```

This will return one of the following strings: "Hello. How are you doing? Speak!" "Hi. How are you doing? Speak!" "Hello. Welcome. Speak!" "Hi. Welcome. Speak!" "Hi my friend."

class alex.components.nlg.template.**TectoTemplateNLG**(*cfg*)
Bases: *alex.components.nlg.template.AbstractTemplateNLG*

Template generation using tecto-trees and NLG rules.

fill_in_template(*tpl*, *svs*)
Filling in tecto-templates, i.e. filling-in strings to templates and using rules to generate the result.

class alex.components.nlg.template.**TemplateNLG**(*cfg*)
Bases: *alex.components.nlg.template.AbstractTemplateNLG*

A simple text-replacement template NLG implementation with the ability to resort to a back-off system if no appropriate template is found.

fill_in_template(*tpl*, *svs*)
Simple text replacement template filling.

Applies template NLG pre- and postprocessing, if applicable.

class alex.components.nlg.template.**TemplateNLGPostprocessing**
Bases: object

Base class for template NLG postprocessing, handles postprocessing of the text resulting from filling in a template.

This base class provides no functionality, it just defines an interface for derived language-specific and/or domain-specific classes.

postprocess(*nlg_text*)

class alex.components.nlg.template.**TemplateNLGPreprocessing**(*ontology*)
Bases: object

Base class for template NLG preprocessing, handles preprocessing of the values to be filled into a template.

This base class provides no functionality, it just defines an interface for derived language-specific and/or domain-specific classes.

preprocess(*svs_dict*)

alex.components.nlg.test_tectotpl module

class alex.components.nlg.test_tectotpl.**TestTectoTemplateNLG**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_tecto_template_nlg()

alex.components.nlg.test_template module

class alex.components.nlg.test_template.**TestTemplateNLG**(*methodName='runTest'*)

Bases: unittest.case.TestCase

setUp()

test_template_nlg()

test_template_nlg_r()

Module contents

alex.components.slu package

Submodules

alex.components.slu.autopath module self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing "pypy" is in sys.path.

If you modify the master "autopath.py" version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

alex.components.slu.base module

class alex.components.slu.base.**CategoryLabelDatabase**(*file_name=None*)

Bases: object

Provides a convenient interface to a database of slot value pairs aka category labels.

Attributes: synonym_value_category: a list of (form, value, category label) tuples

In an utterance:

- there can be multiple surface forms in an utterance
- surface forms can overlap
- a surface form can map to multiple category labels

Then when detecting surface forms / category labels in an utterance:

1. find all existing surface forms / category labels and generate a new utterance with for every found surface form and category label (called abstracted), where the original surface form is replaced by its category label
    - instead of testing all surface forms from the CLDB from the longest to the shortest in the utterance, we test all the substrings in the utterance from the longest to the shortest

form_upnames_vals

list of tuples (form, upnames_vals) from the database where upnames_vals is a dictionary

{name.upper(): all values for this (form, name)}.

form_val_upname
list of tuples (form, value, name.upper()) from the database

gen_form_value_cl_list()
Generates an list of form, value, category label tuples from the database. This list is ordered where the tuples with the longest surface forms are at the beginning of the list.

Returns none

gen_mapping_form2value2cl()
Generates an list of form, value, category label tuples from the database . This list is ordered where the tuples with the longest surface forms are at the beginning of the list.

Returns none

gen_synonym_value_category()

load(*file_name=None*, *db_mod=None*)

normalise_database()
Normalise database. E.g., split utterances into sequences of words.

class alex.components.slu.base.**SLUInterface**(*preprocessing*, *cfg*, **args*, ***kwargs*)
Bases: object

Defines a prototypical interface each SLU parser should provide.

It should be able to parse:

1. **an utterance hypothesis (an instance of UtteranceHyp)**
   - output: an instance of SLUHypothesis
2. **an n-best list of utterances (an instance of UtteranceNBList)**
   - output: an instance of SLUHypothesis
3. **a confusion network (an instance of UtteranceConfusionNetwork)**
   - output: an instance of SLUHypothesis

extract_features(**args*, ***kwargs*)

parse(*obs*, **args*, ***kwargs*)
Check what the input is and parse accordingly.

parse_1_best(*obs*, **args*, ***kwargs*)

parse_confnet(*obs*, *n=40*, **args*, ***kwargs*)
Parses an observation featuring a word confusion network using the parse_nblist method.

Arguments:

obs – a dictionary of observations :: observation type -> observed value where observation type is one of values for ‘obs_type’ used in ‘ft_props’, and observed value is the corresponding observed value for the input

n – depth of the n-best list generated from the confusion network args – further positional arguments that should be passed to the

‘parse_1_best’ method call

kwargs – further keyword arguments that should be passed to the ‘parse_1_best’ method call

parse_nblist(*obs*, **args*, ***kwargs*)
Parses an observation featuring an utterance n-best list using the parse_1_best method.

Arguments:

obs – a dictionary of observations :: observation type -> observed value where observation type is one of values for 'obs_type' used in 'ft_props', and observed value is the corresponding observed value for the input

args – further positional arguments that should be passed to the 'parse_1_best' method call

kwargs – further keyword arguments that should be passed to the 'parse_1_best' method call

print_classifiers(**args*, ***kwargs*)

prune_classifiers(**args*, ***kwargs*)

prune_features(**args*, ***kwargs*)

save_model(**args*, ***kwargs*)

train(**args*, ***kwargs*)

class alex.components.slu.base.**SLUPreprocessing**(*cldb*, *text_normalization=None*)
Bases: object

Implements preprocessing of utterances or utterances and dialogue acts. The main purpose is to replace all values in the database by their category labels (slot names) to reduce the complexity of the input utterances.

In addition, it implements text normalisation for SLU input, e.g. removing filler words such as UHM, UM etc., converting "I'm" into "I am" etc. Some normalisation is hard-coded. However, it can be updated by providing normalisation patterns.

normalise(*utt_hyp*)

normalise_confnet(*confnet*)
Normalises the confnet (the output of an ASR).

E.g., it removes filler words such as UHM, UM, etc., converts "I'm" into "I am", etc.

normalise_nblist(*nblist*)
Normalises the N-best list (the output of an ASR).

Parameters **nblist** –

Returns

normalise_utterance(*utterance*)
Normalises the utterance (the output of an ASR).

E.g., it removes filler words such as UHM, UM, etc., converts "I'm" into "I am", etc.

text_normalization_mapping = [(['erm'], []), (['uhm'], []), (['um'], []), (["i'm"], ['i', 'am']), (['(sil)'], []), (['(%hesita

alex.components.slu.common module

alex.components.slu.common.**get_slu_type**(*cfg*)
Reads the SLU type from the configuration.

alex.components.slu.common.**slu_factory**(*cfg*, *slu_type=None*)
Creates an SLU parser.

Parameters

- **cfg** –
- **slu_type** –
- **require_model** –
- **training** –

- **verbose** –

alex.components.slu.cued_da module

class alex.components.slu.cued_da.**CUEDDialogueAct**(*da_str=None*)

Bases: *alex.components.slu.da.DialogueAct*

CUED-style dialogue act

parse(*da_str*)

class alex.components.slu.cued_da.**CUEDSlot**(*slot_str*)

Bases: object

parse(*slot_str*)

alex.components.slu.cued_da.**load_das**(*das_fname*, *limit=None*, *encoding=u'UTF-8'*)

alex.components.slu.da module

class alex.components.slu.da.**DialogueAct**(*da_str=None*)

Bases: object

Represents a dialogue act (DA), i.e., a set of dialogue act items (DAIs).

The DAIs are stored in the 'dais' attribute, sorted w.r.t. their string representation. This class is not responsible for discarding a DAI which is repeated several times, so that you can obtain a DA that looks like this:

inform(food="chinese")&inform(food="chinese")

Attributes: dais: a list of DAIs that constitute this dialogue act

append(*dai*)

Append a dialogue act item to the current dialogue act.

dais

extend(*dais*)

get_slots_and_values()

Returns all slot names and values in the dialogue act.

has_dat(*dat*)

Checks whether any of the dialogue act items has a specific dialogue act type.

has_only_dat(*dat*)

Checks whether all the dialogue act items has a specific dialogue act type.

merge(*da*)

Merges another DialogueAct into self. This is done by concatenating lists of the DAIs, and sorting and merging own DAIs afterwards.

If sorting is not desired, use 'extend' instead.

merge_same_dais()

Merges same DAIs. I.e., if they are equal on extension but differ in original values, merges the original values together, and keeps the single DAI. This method causes the list of DAIs to be sorted.

parse(*da_str*)

Parses the dialogue act from text.

If any DAIs have been already defined for this DA, they will be overwritten.

sort()

Sorts own DAIs and merges the same ones.

class alex.components.slu.da.**DialogueActConfusionNetwork**
Bases: *alex.components.slu.da.SLUHypothesis*, *alex.ml.hypothesis.ConfusionNetwork*

Dialogue act item confusion network. This is a very simple implementation in which all dialogue act items are assumed to be independent. Therefore, the network stores only posteriors for dialogue act items.

This can be efficiently stored as a list of DAIs each associated with its probability. The alternative for each DAI is that there is no such DAI in the DA. This can be represented as the null() dialogue act and its probability is 1 - p(DAI).

If there are more than one null() DA in the output DA, then they are collapsed into one null() DA since it means the same.

Please note that in the confusion network, the null() dialogue acts are not explicitly modelled.

get_best_da()
Return the best dialogue act (one with the highest probability).

get_best_da_hyp(*use_log=False*, *threshold=None*, *thresholds=None*)
Return the best dialogue act hypothesis.

Arguments:

use_log: whether to express probabilities on the log-scale (otherwise, they vanish easily in a moderately long confnet)

threshold: threshold on probabilities – items with probability exceeding the threshold will be present in the output (default: 0.5)

thresholds: threshold on probabilities – items with probability exceeding the threshold will be present in the output. This is a mapping {dai -> threshold}, and if supplied, overwrites settings of 'threshold'. If not supplied, it is ignored.

get_best_nonnull_da()
Return the best dialogue act (with the highest probability) ignoring the best null() dialogue act item.

Instead of returning the `null()` act, it returns the most probable DAI with a defined slot name.

get_da_nblist(*n=10*, *prune_prob=0.005*)
Parses the input dialogue act item confusion network and generates N-best hypotheses.

The result is a list of dialogue act hypotheses each with a with assigned probability. The list also include a dialogue act for not having the correct dialogue act in the list - other().

Generation of hypotheses will stop when the probability of the hypotheses is smaller then the `prune_prob`.

items()

classmethod make_from_da(*da*)

class alex.components.slu.da.**DialogueActHyp**(*prob=None*, *da=None*)
Bases: *alex.components.slu.da.SLUHypothesis*

Provides functionality of 1-best hypotheses for dialogue acts.

get_best_da()

get_da_nblist()

class alex.components.slu.da.**DialogueActItem**(*dialogue_act_type=None*, *name=None*, *value=None*, *dai=None*, *attrs=None*, *alignment=None*)
Bases: *alex.ml.features.Abstracted*

Represents dialogue act item which is a component of a dialogue act.

Each dialogue act item is composed of

1. dialogue act type - e.g. inform, confirm, request, select, hello
2. **slot name and value pair - e.g. area, pricerange, food for name and** centre, cheap, or Italian for value

Attributes: dat: dialogue act type (a string) name: slot name (a string or None) value: slot value (a string or None)

add_unnorm_value(*newval*)
Registers 'newval' as another alternative unnormalised value for the value of this DAI's slot.

alignment

category_label2value(*catlabs=None*)
Use this method to substitute back the original value for the category label as the value of this DAI.

Arguments:

catlabs: an optional mapping of category labels to tuples (slot value, surface form), as obtained from alex.components.slu:SLUPreprocessing

If this object does not remember its original value, it takes it from the provided mapping.

dat

extension()
Returns an extension of self, i.e., a new DialogueActItem without hidden fields, such as the original value/category label.

get_unnorm_values()
Retrieves the original unnormalised vaues of this DAI.

has_category_label()
whether the current DAI value is the category label

is_null()
whether this object represents the 'null()' DAI

iter_typeval()

merge_unnorm_values(*other*)
Merges unnormalised values of 'other' to unnormalised values of 'self'.

name

normalised2value()
Use this method to substitute back an unnormalised value for the normalised one as the value of this DAI.

Returns True iff substitution took place. Returns False if no more unnormalised values are remembered as a source for the normalised value.

orig_values

parse(*dai_str*)
Parses the dialogue act item in text format into a structured form.

replace_typeval(*orig*, *replacement*)

splitter = u':'

unnorm_values

value

value2category_label(*label=None*)
Use this method to substitute a category label for value of this DAI.

value2normalised(*normalised*)
Use this method to substitute a normalised value for value of this DAI.

class alex.components.slu.da.**DialogueActNBList**
Bases: *alex.components.slu.da.SLUHypothesis*, *alex.ml.hypothesis.NBList*

Provides functionality of N-best lists for dialogue acts.

When updating the N-best list, one should do the following.

1.add DAs or parse a confusion network

2.merge and normalise, in either order

Attributes:

n_best: the list containing pairs [prob, DA] sorted from the most probable to the least probable ones

add_other()

get_best_da()
Returns the most probable dialogue act.

DEPRECATED. Use get_best instead.

get_best_nonnull_da()
Return the best dialogue act (with the highest probability).

get_confnet()

has_dat(*dat*)

merge()
Adds up probabilities for the same hypotheses. Takes care to keep track of original, unnormalised DAI values. Returns self.

normalise()
The N-best list is extended to include the "other()" dialogue act to represent those semantic hypotheses which are not included in the N-best list.

DEPRECATED. Use add_other instead.

scale()
Scales the n-best list to sum to one.

sort()
DEPRECATED, going to be removed.

class alex.components.slu.da.**SLUHypothesis**
Bases: *alex.ml.hypothesis.Hypothesis*

This is the base class for all forms of probabilistic SLU hypotheses representations.

alex.components.slu.da.**load_das**(*das_fname*, *limit=None*, *encoding=u'UTF-8'*)
Loads a dictionary of DAs from a given file.

The file is assumed to contain lines of the following form:

[[:space:]..]<key>[[:space:]..]=>[[:space:]..]<DA>[[:space:]..]

or just (without keys):

[[:space:]..]<DA>[[:space:]..]

Arguments: das_fname – path towards the file to read the DAs from limit – limit on the number of DAs to read encoding – the file encoding

Returns a dictionary with DAs (instances of DialogueAct) as values.

`alex.components.slu.da.`**`merge_slu_confnets`**(*confnet_hyps*)
Merge multiple dialogue act confusion networks.

`alex.components.slu.da.`**`merge_slu_nblists`**(*multiple_nblists*)
Merge multiple dialogue act N-best lists.

`alex.components.slu.da.`**`save_das`**(*file_name*, *das*, *encoding=u'UTF-8'*)

alex.components.slu.dailrclassifier module This is a rewrite of the DAILogRegClassifier `from dailrclassifier_old.py`. The underlying approach is the same; however, the way how the features are computed is changed significantly.

class `alex.components.slu.dailrclassifier.`**`DAILogRegClassifier`**(*cldb*, *preprocessing*, *features_size=4*, **args*, ***kwargs*)

Bases: *alex.components.slu.base.SLUInterface*

Implements learning of dialogue act item classifiers based on logistic regression.

The parser implements a parser based on set of classifiers for each dialogue act item. When parsing the input utterance, the parse classifies whether a given dialogue act item is present. Then, the output dialogue act is composed of all detected dialogue act items.

Dialogue act is defined as a composition of dialogue act items. E.g.

confirm(drinks="wine")&inform(name="kings shilling") <=> 'does kings serve wine'

where confirm(drinks="wine") and inform(name="kings shilling") are two dialogue act items.

This parser uses logistic regression as the classifier of the dialogue act items.

`abstract_utterance`(*utterance*)
Return a list of possible abstractions of the utterance.

Parameters **`utterance`** – an Utterance instance

Returns a list of abstracted utterance, form, value, category label tuples

`extract_classifiers`(*das*, *utterances*, *verbose=False*)

`gen_classifiers_data`(*min_pos_feature_count=5*, *min_neg_feature_count=5*, *verbose=False*, *verbose2=False*)

`get_abstract_da`(*da*, *fvcs*)

`get_abstract_utterance`(*utterance*, *fvc*)
Return an utterance with the form inn fvc abstracted to its category label

Parameters

- **`utterance`** – an Utterance instance
- **`fvc`** – a form, value, category label tuple

Returns return the abstracted utterance

`get_abstract_utterance2`(*utterance*)
Return an utterance with the form un fvc abstracted to its category label

Parameters **`utterance`** – an Utterance instance

Returns return the abstracted utterance

get_features(*obs*, *fvc*, *fvcs*)
Generate utterance features for a specific utterance given by utt_idx.

Parameters

- **obs** – the utterance being processed in multiple formats
- **fvc** – a form, value category tuple describing how the utterance should be abstracted

Returns a set of features from the utterance

get_features_in_confnet(*confnet*, *fvc*, *fvcs*)

get_features_in_nblist(*nblist*, *fvc*, *fvcs*)

get_features_in_utterance(*utterance*, *fvc*, *fvcs*)
Returns features extracted from the utterance observation. At this moment, the function extracts N-grams of size self.feature_size. These N-grams are extracted from:

•the original utterance,

•the abstracted utterance for the given FVC

•the abstracted where all other FVCs are abstracted as well

Parameters

- **utterance** –
- **fvc** –

Returns the UtteranceFeatures instance

get_fvc(**args*, ***kwds*)
This function returns the form, value, category label tuple for any of the following classses

•Utterance

•UttranceNBList

•UtteranceConfusionNetwork

Parameters **obs** – the utterance being processed in multiple formats

Returns a list of form, value, and category label tuples found in the input sentence

get_fvc_in_confnet(*confnet*)
Return a list of all form, value, category label tuples in the confusion network.

Parameters **nblist** – an UtteranceConfusionNetwork instance

Returns a list of form, value, and category label tuples found in the input sentence

get_fvc_in_nblist(*nblist*)
Return a list of all form, value, category label tuples in the nblist.

Parameters **nblist** – an UtteranceNBList instance

Returns a list of form, value, and category label tuples found in the input sentence

get_fvc_in_utterance(*utterance*)
Return a list of all form, value, category label tuples in the utterance. This is useful to find/guess what category label level classifiers will be necessary to instantiate.

Parameters **utterance** – an Utterance instance

Returns a list of form, value, and category label tuples found in the input sentence

load_model(*file_name*)

parse_1_best(*obs={}*, *ret_cl_map=False*, *verbose=False*, **args*, ***kwargs*)
Parse `utterance` and generate the best interpretation in the form of a dialogue act (an instance of DialogueAct).

The result is the dialogue act confusion network.

parse_X(*utterance*, *verbose=False*)

parse_confnet(*obs*, *verbose=False*, **args*, ***kwargs*)
Parses the word confusion network by generating an n-best list and parsing this n-best list.

parse_nblist(*obs*, *verbose=False*, **args*, ***kwargs*)
Parses n-best list by parsing each item on the list and then merging the results.

print_classifiers()

prune_classifiers(*min_classifier_count=5*)

prune_features(*clser*, *min_pos_feature_count*, *min_neg_feature_count*, *verbose=False*)

save_model(*file_name*, *gzip=None*)

train(*inverse_regularisation=1.0*, *verbose=True*)

class `alex.components.slu.dailrclassifier.`**Features**
Bases: `object`

This is a simple feature object. It is a light version of an unnecessary complicated alex.ml.features.Features class.

get_feature_vector(*features_mapping*)

get_feature_vector_lil(*features_mapping*)

merge(*features*, *weight=1.0*, *prefix=None*)
Merges passed feature dictionary with its own features. To the features can be applied weight factor or the features can be added as a binary feature. If a prefix is provided, then the features are added with the prefixed feature name.

Parameters

- **features** – a dictionary-like object with features as keys and values
- **weight** – a weight of added features with respect to already existing features. If None, then it is is added as a binary feature
- **prefix** – prefix for a name of an added features, This is useful when one want to distinguish between similarly generated features

prune(*remove_features*)
Prune all features in the `remove_feature` set.

Parameters **remove_features** – a set of features to be pruned.

scale(*scale=1.0*)
Scale all features with the scale.

Parameters **scale** – the scale factor.

class alex.components.slu.dailrclassifier.**UtteranceFeatures**(*type=u'ngram'*, *size=3*, *utterance=None*)

Bases: *alex.components.slu.dailrclassifier.Features*

This is a simple feature object. It is a light version of a alex.components.asr.utterance.UtteranceFeatures class.

parse(*utt*)

alex.components.slu.dainnclassifier module

alex.components.slu.exceptions module

exception alex.components.slu.exceptions.**CuedDialogueActError**

Bases: *alex.components.slu.exceptions.SLUException*

exception alex.components.slu.exceptions.**DAIKernelException**

Bases: *alex.components.slu.exceptions.SLUException*

exception alex.components.slu.exceptions.**DAILRException**

Bases: *alex.components.slu.exceptions.SLUException*

exception alex.components.slu.exceptions.**DialogueActConfusionNetworkException**

Bases: *alex.components.slu.exceptions.SLUException*, *alex.ml.hypothesis.ConfusionNetworkExc*

exception alex.components.slu.exceptions.**DialogueActException**

Bases: *alex.components.slu.exceptions.SLUException*

exception alex.components.slu.exceptions.**DialogueActItemException**

Bases: *alex.components.slu.exceptions.SLUException*

exception alex.components.slu.exceptions.**DialogueActNBListException**

Bases: *alex.components.slu.exceptions.SLUException*

exception alex.components.slu.exceptions.**SLUConfigurationException**

Bases: *alex.components.slu.exceptions.SLUException*

exception alex.components.slu.exceptions.**SLUException**

Bases: *alex.AlexException*

alex.components.slu.templateclassifier module

class alex.components.slu.templateclassifier.**TemplateClassifier**(*config*)

Bases: object

This parser is based on matching examples of utterances with known semantics against input utterance. The semantics of the example utterance which is closest to the input utterance is provided as a output semantics.

"Hi" => hello() "I can you give me a phone number" => request(phone) "I would like to have a phone number please" => request(phone)

The first match is reported as the resulting dialogue act.

parse(*asr_hyp*)

readRules(*file_name*)

alex.components.slu.test_da module

class alex.components.slu.test_da.**TestDA**(*methodName='runTest'*)

Bases: unittest.case.TestCase

test_merge_slu_confnets()

test_merge_slu_nblists_full_nbest_lists()

test_swapping_merge_normalise()

class alex.components.slu.test_da.**TestDialogueActConfusionNetwork**(*methodName='runTest'*)

Bases: unittest.case.TestCase

test_add_merge()

test_get_best_da()

test_get_best_da_hyp()

test_get_best_nonnull_da()

test_get_da_nblist()

test_get_prob()

test_make_from_da()

test_merge()

test_normalise()

test_prune()

test_sort()

alex.components.slu.test_dailrclassifier module

class alex.components.slu.test_dailrclassifier.**TestDAILogRegClassifier**(*methodName='runTest'*)

Bases: unittest.case.TestCase

test_parse_X()

alex.components.slu.test_dainnclassifier module

class alex.components.slu.test_dainnclassifier.**TestDAINNClassifier**(*methodName='runTest'*)

Bases: unittest.case.TestCase

setUp()

tearDown()

test_parse_X()

Module contents

alex.components.tts package

Submodules

alex.components.tts.autopath module self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing "pypy" is in sys.path.

If you modify the master "autopath.py" version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

alex.components.tts.base module

class alex.components.tts.base.**TTSInterface**(*cfg*)

Bases: object

synthesize(*text*)

alex.components.tts.common module

alex.components.tts.exceptions module

exception alex.components.tts.exceptions.**TTSException**

Bases: *alex.AlexException*

alex.components.tts.flite module

alex.components.tts.google module

alex.components.tts.preprocessing module

class alex.components.tts.preprocessing.**TTSPreprocessing**(*cfg*, *file_name*)

Bases: object

Preprocess words that are hard to pronounce for the current TTS engine.

load(*file_name*)

process(*text*)

Applies all substitutions on the input text and returns the result.

class alex.components.tts.preprocessing.**TTSPreprocessingException**

Bases: object

alex.components.tts.speechtech module

alex.components.tts.test_google module

alex.components.tts.test_voicerss module

alex.components.tts.voicerss module

Module contents

alex.components.vad package

Submodules

alex.components.vad.ffnn module

alex.components.vad.gmm module

class alex.components.vad.gmm.**GMMVAD**(*cfg*)

This is implementation of a GMM based voice activity detector.

It only implements decisions whether input frame is speech of non speech. It returns the posterior probability of speech for N last input frames.

decide(*data*)

Processes the input frame whether the input segment is speech or non speech.

The returned values can be in range from 0.0 to 1.0. It returns 1.0 for 100% speech segment and 0.0 for 100% non speech segment.

alex.components.vad.power module

class alex.components.vad.power.**PowerVAD**(*cfg*)

This is implementation of a simple power based voice activity detector.

It only implements simple decisions whether input frame is speech of non speech.

decide(*frame*)

Returns whether the input segment is speech or non speech.

The returned values can be in range from 0.0 to 1.0. It returns 1.0 for 100% speech segment and 0.0 for 100% non speech segment.

Module contents

Module contents

alex.corpustools package

Submodules

alex.corpustools.asr_decode module

alex.corpustools.asrscore module

alex.corpustools.asrscore.**score**(*fn_reftext*, *fn_testtext*, *outfile=<open file '<stdout>', mode 'w'>*)

alex.corpustools.asrscore.**score_file**(*reftext*, *testtext*)

Computes ASR scores between reference and test word strings.

Parameters

- **reftext** –
- **testtext** –

Returns a tuple with percentages of correct, substitutions, deletions, insertions, error rate, and a number of reference words.

alex.corpustools.autopath module

self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing "pypy" is in sys.path.

If you modify the master "autopath.py" version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

alex.corpustools.cued-audio2ufal-audio module

alex.corpustools.cued-call-logs-sem2ufal-call-logs-sem module

alex.corpustools.cued-sem2ufal-sem module

alex.corpustools.cued module

This module is meant to collect functionality for handling call logs – both working with the call log files in the filesystem, and parsing them.

`alex.corpustools.cued.`**`find_logs`**(*infname*, *ignore_list_file=None*, *verbose=False*)

Finds CUED logs below the paths specified and returns their filenames. The logs are determined as files matching one of the following patterns:

user-transcription.norm.xml user-transcription.xml user-transcription-all.xml

If multiple patterns are matched by files in the same directory, only the first match is taken.

Arguments:

infname – either a directory, or a file. In the first case, logs are looked for below that directory. In the latter case, the file is read line by line, each line specifying a directory or a glob determining the log to include.

ignore_list_file – a file of absolute paths or globs (can be mixed) specifying logs that should be excluded from the results

verbose – print lots of output?

Returns a set of paths to files satisfying the criteria.

`alex.corpustools.cued.`**`find_wavs`**(*infname*, *ignore_list_file=None*)

Finds wavs below the paths specified and returns their filenames.

Arguments:

infname – either a directory, or a file. In the first case, wavs are looked for below that directory. In the latter case, the file is read line by line, each line specifying a directory or a glob determining the wav to include.

ignore_list_file – a file of absolute paths or globs (can be mixed) specifying wavs that should be excluded from the results

Returns a set of paths to files satisfying the criteria.

alex.corpustools.cued.**find_with_ignorelist**(*infname*, *pat*, *ignore_list_file=None*, *find_kwargs={}*)

Finds specific files below the paths specified and returns their filenames.

Arguments: pat – globbing pattern specifying the files to look for infname – either a directory, or a file. In the first case, wavs are

looked for below that directory. In the latter case, the file is read line by line, each line specifying a directory or a glob determining the wav to include.

ignore_list_file – a file of absolute paths or globs (can be mixed) specifying wavs that should be excluded from the results

find_kwargs – if provided, this dictionary is used as additional keyword arguments for the function 'utils.fs.find' for finding positive examples of files (not the ignored ones)

Returns a set of paths to files satisfying the criteria.

alex.corpustools.cued2utt_da_pairs module

class alex.corpustools.cued2utt_da_pairs.**TurnRecord**(*transcription*, *cued_da*, *cued_dahyp*, *asrhyp*, *audio*)

Bases: tuple

asrhyp
Alias for field number 3

audio
Alias for field number 4

cued_da
Alias for field number 1

cued_dahyp
Alias for field number 2

transcription
Alias for field number 0

alex.corpustools.cued2utt_da_pairs.**extract_trns_sems**(*infname*, *verbose*, *fields=None*, *ignore_list_file=None*, *do_exclude=True*, *normalise=True*, *known_words=None*)

Extracts transcriptions and their semantic annotation from a directory containing CUED call log files.

Arguments:

infname – either a directory, or a file. In the first case, logs are looked for below that directory. In the latter case, the file is read line by line, each line specifying a directory or a glob determining the call log to include.

verbose – print lots of output? fields – names of fields that should be required for the output.

Field names are strings corresponding to the element names in the transcription XML format. (default: all five of them)

ignore_list_file – a file of absolute paths or globs (can be mixed) specifying logs that should be skipped

normalise – whether to do normalisation on transcriptions do_exclude – whether to exclude transcriptions not considered suitable known_words – a collection of words. If provided, transcriptions are

excluded which contain other words. If not provided, excluded are transcriptions that contain any of _excluded_characters. What "excluded" means depends on whether the transcriptions are required by being specified in 'fields'.

Returns a list of TurnRecords.

alex.corpustools.cued2utt_da_pairs.**extract_trns_sems_from_file**(*fname*, *verbose*, *fields=None*, *normalise=True*, *do_exclude=True*, *known_words=None*, *robust=False*)

Extracts transcriptions and their semantic annotation from a CUED call log file.

Arguments: fname – path towards the call log file verbose – print lots of output? fields – names of fields that should be required for the output.

Field names are strings corresponding to the element names in the transcription XML format. (default: all five of them)

normalise – whether to do normalisation on transcriptions do_exclude – whether to exclude transcriptions not considered suitable known_words – a collection of words. If provided, transcriptions are

excluded which contain other words. If not provided, excluded are transcriptions that contain any of _excluded_characters. What "excluded" means depends on whether the transcriptions are required by being specified in 'fields'.

robust – whether to assign recordings to turns robustly or trust where they are in the log. This could be useful for older CUED logs where the elements sometimes escape to another <turn> than they belong. However, in cases where 'robust' leads to finding the correct recording for the user turn, the log is damaged at other places too, and the resulting turn record would be misleading. Therefore, we recommend leaving robust=False.

Returns a list of TurnRecords.

alex.corpustools.cued2utt_da_pairs.**write_asrhyp_sem**(*outdir*, *fname*, *data*)

alex.corpustools.cued2utt_da_pairs.**write_asrhyp_semhyp**(*outdir*, *fname*, *data*)

alex.corpustools.cued2utt_da_pairs.**write_data**(*outdir*, *fname*, *data*, *tpt*)

alex.corpustools.cued2utt_da_pairs.**write_trns_sem**(*outdir*, *fname*, *data*)

alex.corpustools.cued2wavaskey module

Finds CUED XML files describing calls in the directory specified, extracts a couple of fields from them for each turn (transcription, ASR 1-best, semantics transcription, SLU 1-best) and outputs them to separate files in the following format:

{wav_filename} => {field}

An example ignore list file could contain the following three lines:

/some-path/call-logs/log_dir/some_id.wav some_id.wav jurcic-??[13579]*.wav

The first one is an example of an ignored path. On UNIX, it has to start with a slash. On other platforms, an analogic convention has to be used.

The second one is an example of a literal glob.

The last one is an example of a more advanced glob. It says basically that all odd dialogue turns should be ignored.

alex.corpustools.cued2wavaskey.**main**(*args*)

alex.corpustools.cuedda module

class alex.corpustools.cuedda.**CUEDDialogueAct**(*text*, *da*, *database=None*, *dictionary=None*)

get_cued_da()

get_slots_and_values()

get_ufal_da()

parse()

class alex.corpustools.cuedda.**CUEDSlot**(*slot*)

parse()

alex.corpustools.fisherptwo2ufal-audio module

alex.corpustools.grammar_weighted module

class alex.corpustools.grammar_weighted.**A**(**rules*)
Bases: *alex.corpustools.grammar_weighted.Alternative*

class alex.corpustools.grammar_weighted.**Alternative**(**rules*)
Bases: *alex.corpustools.grammar_weighted.Rule*

sample()

class alex.corpustools.grammar_weighted.**GrammarGen**(*root*)
Bases: object

sample(*n*)
Sampling of n sentences.

sample_uniq(*n*)
Unique sampling of n sentences.

class alex.corpustools.grammar_weighted.**O**(*rule*, *prob=0.5*)
Bases: *alex.corpustools.grammar_weighted.Option*

class alex.corpustools.grammar_weighted.**Option**(*rule*, *prob=0.5*)
Bases: *alex.corpustools.grammar_weighted.Rule*

sample()

class alex.corpustools.grammar_weighted.**Rule**
Bases: object

class alex.corpustools.grammar_weighted.**S**(**rules*)
Bases: *alex.corpustools.grammar_weighted.Sequence*

class alex.corpustools.grammar_weighted.**Sequence**(**rules*)
Bases: *alex.corpustools.grammar_weighted.Rule*

sample()

class alex.corpustools.grammar_weighted.**T**(*string*)
Bases: *alex.corpustools.grammar_weighted.Terminal*

class alex.corpustools.grammar_weighted.**Terminal**(*string*)
Bases: *alex.corpustools.grammar_weighted.Rule*

sample()

class alex.corpustools.grammar_weighted.**UA**(**rules*)
Bases: *alex.corpustools.grammar_weighted.UniformAlternative*

class alex.corpustools.grammar_weighted.**UniformAlternative**(**rules*)
Bases: *alex.corpustools.grammar_weighted.Rule*

load(*fn*)
Load alternative terminal strings from a file.

Parameters **fn** – a file name

sample()

alex.corpustools.grammar_weighted.**as_terminal**(*rule*)

alex.corpustools.grammar_weighted.**as_weight_tuple**(*rule*, *def_weight=1.0*)

alex.corpustools.grammar_weighted.**clamp_01**(*number*)

alex.corpustools.grammar_weighted.**counter_weight**(*rules*)

alex.corpustools.grammar_weighted.**remove_spaces**(*utterance*)

alex.corpustools.librispeech2ufal-audio module

alex.corpustools.lm module

alex.corpustools.malach-en2ufal-audio module

alex.corpustools.merge_uttcns module

alex.corpustools.merge_uttcns.**find_best_cn**(*cns*)
Determines which one of decoded confnets seems the best.

alex.corpustools.merge_uttcns.**merge_files**(*fnames*, *outfname*)

alex.corpustools.num_time_stats module

Traverses the filesystem below a specified directory, looking for call log directories. Writes a file containing statistics about each phone number (extracted from the call log dirs' names):

- number of calls
- total size of recorded wav files
- last expected date the caller would call
- last date the caller actually called
- the phone number

Call with -h to obtain the help for command line arguments.

2012-12-11 Matěj Korvas

alex.corpustools.num_time_stats.**get_call_data_from_fs**(*rootdir*)

alex.corpustools.num_time_stats.**get_call_data_from_log**(*log_fname*)

alex.corpustools.num_time_stats.**get_timestamp**(*date*)
Total seconds in the timedelta.

alex.corpustools.num_time_stats.**mean**(*collection*)

alex.corpustools.num_time_stats.**sd**(*collection*)

alex.corpustools.num_time_stats.**set_and_ret**(*indexable*, *idx*, *val*)

alex.corpustools.num_time_stats.**var**(*collection*)

alex.corpustools.recording_splitter module

alex.corpustools.semscore module

alex.corpustools.semscore.**load_semantics**(*file_name*)

alex.corpustools.semscore.**score**(*fn_refsem*, *fn_testsem*, *item_level=False*, *detailed_error_output=False*, *outfile=<open file '<stdout>', mode 'w'>*)

alex.corpustools.semscore.**score_da**(*ref_da*, *test_da*, *daid*)
Computed according to http://en.wikipedia.org/wiki/Precision_and_recall

alex.corpustools.semscore.**score_file**(*refsem*, *testsem*)

alex.corpustools.split-asr-data module

alex.corpustools.srilm_ppl_filter module

alex.corpustools.srilm_ppl_filter.**main**()

alex.corpustools.srilm_ppl_filter.**srilm_scores**(*d3*)

alex.corpustools.text_norm_cs module

This module provides tools for **CZECH** normalisation of transcriptions, mainly for those obtained from human transcribers.

alex.corpustools.text_norm_cs.**normalise_text**(*text*)
Normalises the transcription. This is the main function of this module.

alex.corpustools.text_norm_cs.**exclude_by_dict**(*text*, *known_words*)
Determines whether text is not good enough and should be excluded.

"Good enough" is defined as having all its words present in the 'known_words' collection.

alex.corpustools.text_norm_en module

This module provides tools for **ENGLISH** normalisation of transcriptions, mainly for those obtained from human transcribers.

alex.corpustools.text_norm_en.**normalise_text**(*text*)
Normalises the transcription. This is the main function of this module.

alex.corpustools.text_norm_en.**exclude_by_dict**(*text*, *known_words*)
Determines whether text is not good enough and should be excluded.

"Good enough" is defined as having all its words present in the 'known_words' collection.

alex.corpustools.text_norm_es module

This module provides tools for **ENGLISH** normalisation of transcriptions, mainly for those obtained from human transcribers.

alex.corpustools.text_norm_es.**normalise_text**(*text*)
Normalises the transcription. This is the main function of this module.

alex.corpustools.text_norm_es.**exclude_by_dict**(*text*, *known_words*)
Determines whether text is not good enough and should be excluded.

"Good enough" is defined as having all its words present in the 'known_words' collection.

alex.corpustools.ufal-call-logs-audio2ufal-audio module

alex.corpustools.ufal-transcriber2ufal-audio module

alex.corpustools.ufaldatabase module

alex.corpustools.ufaldatabase.**save_database**(*odir*, *slots*)

alex.corpustools.vad-mlf-from-ufal-audio module

alex.corpustools.voxforge2ufal-audio module

alex.corpustools.wavaskey module

alex.corpustools.wavaskey.**load_wavaskey**(*fname*, *constructor*, *limit=None*, *encoding=u'UTF-8'*)
Loads a dictionary of objects stored in the "wav as key" format.

The input file is assumed to contain lines of the following form:

[[:space:]..]<key>[[:space:]..]=>[[:space:]..]<obj_str>[[:space:]..]

or just (without keys):

[[:space:]..]<obj_str>[[:space:]..]

where <obj_str> is to be given as the only argument to the 'constructor' when constructing the objects stored in the file.

Arguments: fname – path towards the file to read the objects from constructor – function that will be called on each string stored in

the file and whose result will become a value of the returned dictionary

limit – limit on the number of objects to read encoding – the file encoding

Returns a dictionary with objects constructed by ‘constructor’ as values.

`alex.corpustools.wavaskey.`**`save_wavaskey`**(*fname*, *in_dict*, *encoding=u'UTF-8'*, *trans=<function <lambda>>*)

Saves a dictionary of objects in the wave as key format into a file.

Parameters

- **`file_name`** – name of the target file
- **`utt`** – a dictionary with the objects where the keys are the names of teh corresponding wave files

Parma trans a function which can transform a saved object

Returns None

Module contents

alex.ml package

Subpackages

alex.ml.bn package

Submodules

alex.ml.bn.autopath module self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing “pypy” is in sys.path.

If you modify the master “autopath.py” version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

alex.ml.bn.factor module This module implements a factor class, which can be used to do computations with probability distributions.

class `alex.ml.bn.factor.`**`Factor`**(*variables*, *variable_values*, *prob_table*, *logarithmetic=True*)

Bases: `object`

Basic factor.

marginalize(*keep*)
Marginalize all but specified variables.

Marginalizing means summing out values which are not in *keep*. The result is a new factor, which contains only variables from *keep*.

Example:

```
>>> f = Factor(['A', 'B'],
...                     {'A': ['a1', 'a2'], 'B': ['b1', 'b2']},
...                     {
...                           ('a1', 'b1'): 0.8,
...                           ('a2', 'b1'): 0.2,
...                           ('a1', 'b2'): 0.3,
...                           ('a2', 'b2'): 0.7
...                     })
>>> result = f.marginalize(['A'])
>>> print result.pretty_print(width=30)
------------------------------
       A            Value
------------------------------
      a1             1.1
      a2             0.9
------------------------------
```

Parameters **keep** (*list of str*) – Variables which should be left in marginalized factor.

Returns Marginalized factor.

Return type *Factor*

most_probable(*n=None*)
Return a list of most probable assignments from the table.

Returns a sorted list of assignment and their values according to their probability. The size of the list can be changed by specifying n.

Parameters **n** (*int*) – The number of most probable elements, which should be returned.

Returns A list of tuples (assignment, value) in descending order.

Return type list of (tuple, float)

normalize(*parents=None*)
Normalize a factor table.

The table is normalized so all elements sum to one. The parents argument is a list of names of parents. If it is specified, then only those rows in table, which share the same parents, are normalized.

Example:

```
>>> f = Factor(['A', 'B'],
...                     {'A': ['a1', 'a2'], 'B': ['b1', 'b2']},
...                     {
...                           ('a1', 'b1'): 3,
...                           ('a1', 'b2'): 1,
...                           ('a2', 'b1'): 1,
...                           ('a2', 'b2'): 1,
...                     })
>>> f.normalize(parents=['B'])
>>> print f.pretty_print(width=30)
------------------------------
```

```
        A          B        Value
------------------------------
        a1         b1        0.75
        a1         b2        0.5
        a2         b1        0.25
        a2         b2        0.5
------------------------------
```

Parameters **parents** (*list*) – Parents of the factor.

observed(*assignment_dict*)
Set observation.

Example:

```
>>> f = Factor(
...     ['X'],
...     {
...         'X': ['x0', 'x1'],
...     },
...     {
...         ('x0',): 0.5,
...         ('x1',): 0.5,
...     })
>>> print f.pretty_print(width=30, precision=3)
------------------------------
       X            Value
------------------------------
      x0             0.5
      x1             0.5
------------------------------
>>> f.observed({('x0',): 0.8, ('x1',): 0.2})
>>> print f.pretty_print(width=30, precision=3)
------------------------------
       X            Value
------------------------------
      x0             0.8
      x1             0.2
------------------------------
```

Parameters **assignment_dict** (*dict or None*) – Observed values for different assignments of values or None.

pretty_print(*width=79*, *precision=10*)
Create a readable representation of the factor.

Creates a table with a column for each variable and value. Every row represents one assignemnt and its corresponding value. The default width of the table is 79 chars, to fit to terminal window.

Parameters

- **width** (*int*) – Width of the table.
- **precision** (*int*) – Precision of values.

Returns Pretty printed factor table.

Return type str

rename_variables(*mapping*)

sum_other()

exception alex.ml.bn.factor.**FactorError**
Bases: exceptions.Exception

alex.ml.bn.factor.**from_log**(*n*)
Convert number from log arithmetic.

Parameters **n** (*number or array like*) – Number to be converted from log arithmetic.

Returns Number in decimal scale.

Return type number or array like

alex.ml.bn.factor.**to_log**(*n*, *out=None*)
Convert number to log arithmetic.

We want to be able to represent zero, therefore every number smaller than epsilon is considered a zero.

Parameters

- **n** (*number or array like*) – Number to be converted.
- **out** (*ndarray*) – Output array.

Returns Number in log arithmetic.

Return type number or array like

alex.ml.bn.lbp module Belief propagation algorithms for factor graph.

class alex.ml.bn.lbp.**BP**
Bases: object

Abstract class for Belief Propagation algorithm.

run()
Run inference algorithm.

exception alex.ml.bn.lbp.**BPError**
Bases: exceptions.Exception

class alex.ml.bn.lbp.**LBP**(*strategy='sequential'*, ***kwargs*)
Bases: *alex.ml.bn.lbp.BP*

Loopy Belief Propagation.

LBP is an approximative inference algorithm for factor graphs. LBP works with generic factor graphs. It does accurate inference for trees and is equal to sum-product algorithm there.

It is possible to specify which strategy should be used for choosing next node for update. Sequential strategy will update nodes in exact order in which they were added. Tree strategy will assume the graph is a tree (without checking) and will do one pass of sum-product algorithm.

add_layer(*layer*)

add_layers(*layers*)
Add layers of nodes to graph.

add_nodes(*nodes*)
Add nodes to graph.

clear_layers()

clear_nodes()

init_messages()

run(*n_iterations=1*, *from_layer=None*)
Run the lbp algorithm.

exception alex.ml.bn.lbp.**LBPError**
Bases: *alex.ml.bn.lbp.BPError*

alex.ml.bn.node module Node representations for factor graph.

class alex.ml.bn.node.**DirichletFactorNode**(*name*, *aliases=None*)
Bases: *alex.ml.bn.node.FactorNode*

Node containing dirichlet factor.

add_neighbor(*node*, *parent=True*, ***kwargs*)

init_messages()

message_from(*node*, *message*)

message_to(*node*)

normalize(*parents=None*)

update()

class alex.ml.bn.node.**DirichletParameterNode**(*name*, *alpha*, *aliases=None*)
Bases: *alex.ml.bn.node.VariableNode*

Node containing parameter.

add_neighbor(*node*)

init_messages()

message_from(*node*, *message*)

message_to(*node*)

normalize(*parents=None*)

update()

class alex.ml.bn.node.**DiscreteFactorNode**(*name*, *factor*)
Bases: *alex.ml.bn.node.FactorNode*

Node containing factor.

add_neighbor(*node*, ***kwargs*)

init_messages()

message_from(*node*, *message*)

message_to(*node*)

update()

class alex.ml.bn.node.**DiscreteVariableNode**(*name*, *values*, *logarithmetic=True*)
Bases: *alex.ml.bn.node.VariableNode*

Node containing variable.

add_neighbor(*node*, ***kwargs*)

init_messages()

message_from(*node*, *message*)

message_to(*node*)

most_probable(*n=None*)

observed(*assignment_dict*)
Set observation.

update()

class alex.ml.bn.node.**FactorNode**(*name*, *aliases=None*)
Bases: *alex.ml.bn.node.Node*

exception alex.ml.bn.node.**IncompatibleNeighborError**
Bases: *alex.ml.bn.node.NodeError*

class alex.ml.bn.node.**Node**(*name*, *aliases=None*)
Bases: object

Abstract class for nodes in factor graph.

add_neighbor(*node*)

connect(*node*, ***kwargs*)
Add a neighboring node.

init_messages()

message_from(*node*, *message*)
Save message from neighboring node.

message_to(*node*)
Compute a message to neighboring node.

normalize(*parents=None*)
Normalize belief state.

rename_msg(*msg*)

send_messages()
Send messages to all neighboring nodes.

update()
Update belief state.

exception alex.ml.bn.node.**NodeError**
Bases: exceptions.Exception

class alex.ml.bn.node.**VariableNode**(*name*, *aliases=None*)
Bases: *alex.ml.bn.node.Node*

alex.ml.bn.test_factor module

class alex.ml.bn.test_factor.**TestFactor**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_add()

test_alphas()

test_apply_op_different()

test_apply_op_same()

test_apply_op_scalar()

test_division()

test_expected_value_squared()

test_fast_div()

test_fast_mul()

test_fast_mul_correct()

test_get_assignment_from_index()

test_get_index_from_assignment()

test_logsubexp()

test_marginalize()

test_mul_div()

test_multiplication()

test_multiplication_different_values()

test_observations()

test_parents_normalize()

test_power()

test_rename()

test_setitem()

test_strides()

test_sum_other()

alex.ml.bn.test_lbp module

class alex.ml.bn.test_lbp.**TestLBP**(*methodName='runTest'*)

Bases: unittest.case.TestCase

test_ep()

test_ep_tight()

test_layers()

test_network()

test_single_linked()

alex.ml.bn.test_node module

class alex.ml.bn.test_node.**TestNode**(*methodName='runTest'*)

Bases: unittest.case.TestCase

assertClose(*first*, *second*, *epsilon=1e-06*)

test_dir_tight()

test_network()

test_observed_complex()

test_parameter()

test_parameter_simple()

test_two_factors_one_theta()

test_two_factors_one_theta2()

alex.ml.bn.test_node.**same_or_different**(*assignment*)

alex.ml.bn.utils module

alex.ml.bn.utils.**constant_factor**(*variables*, *variables_dict*, *length*, *logarithmetic=True*)

alex.ml.bn.utils.**constant_factory**(*value*)

Create function returning constant value.

Module contents

alex.ml.ep package

Submodules

alex.ml.ep.node module

class alex.ml.ep.node.**ConstChangeGoal**(*name*, *desc*, *card*, *parameters*, *parents=None*)

Bases: *alex.ml.ep.node.GroupingGoal*

ConstChangeGoal implements all functionality as is include in GroupingGoal; however, it that there are only two transition probabilites for transitions between the same values and the different values.

update()

This function update belief for the goal.

class alex.ml.ep.node.**Goal**(*name*, *desc*, *card*, *parameters*, *parents=None*)

Bases: *alex.ml.ep.node.Node*

Goal can contain only the same values as the observations.

As a consequence, it can contain values of its previous node.

probTable(*value*, *parents*)

This function defines how the coditional probability is computed.

pRemebering - probability that the previous value is correct pObserving - probability that the observed value is correct

setParents(*parents*)

setValues()

The function copy values from its previous node and from observation nodes.

update()

This function update belief for the goal.

class alex.ml.ep.node.**GroupingGoal**(*name*, *desc*, *card*, *parameters*, *parents=None*)

Bases: *alex.ml.ep.node.GroupingNode*, *alex.ml.ep.node.Goal*

GroupingGoal implements all functionality as is include in Goal; however, it only update the values for which was observed some evidence.

setValues()

The function copy values from its previous node and from observation nodes.

update()

This function update belief for the goal.

class alex.ml.ep.node.**GroupingNode**(*name*, *desc*, *card*)
Bases: *alex.ml.ep.node.Node*

addOthers(*value*, *probability*)

explain(*full=None*)
This function explains the values for this node.

In additon to the Node's function, it prints the cardinality of the others set.

splitOff(*value*)
This function split off the value from the others set and place it into the values dict.

class alex.ml.ep.node.**Node**(*name*, *desc*, *card*)
Bases: object

A base class for all nodes in a belief state.

explain(*full=None*)
This function prints the values and their probailities for this node.

getMostProbableValue()
The function returns the most probable value and its probability in a tuple.

getTwoMostProbableValues()
This function returns two most probable values and their probabilities.

The function returns a tuple consisting of two tuples (value, probability).

normalise()
This function normlize the sum of all probabilities to 1.0

alex.ml.ep.test module

alex.ml.ep.test.**random**() → x in the interval [0, 1).

alex.ml.ep.turn module

class alex.ml.ep.turn.**Turn**

Module contents

alex.ml.gmm package

Submodules

alex.ml.gmm.gmm module

class alex.ml.gmm.gmm.**GMM**(*n_features=1*, *n_components=1*, *thresh=0.001*, *min_covar=0.001*, *n_iter=1*)
This is a GMM model of the input data. It is memory efficient so that it can process very large input array like objects.

The mixtures are incrementally added by splitting the heaviest component in two components and perturbation of the original mean.

expectation(*x*)
Evaluate one example

fit(*X*)

load_model(*file_name*)
Load the model from a pickle.load

log_multivariate_normal_density_diag(*x*, *means=0.0*, *covars=1.0*)
Compute Gaussian log-density at X for a diagonal model

mixup(*n_new_mixies*)
Add n new mixies to the mixture.

save_model(*file_name*)
Save the GMM model as a pickle.

score(*x*)
Get the log prob of the x variable being generated by the mixture.

Module contents

class alex.ml.gmm.**GMM**(*n_features=1*, *n_components=1*, *thresh=0.001*, *min_covar=0.001*, *n_iter=1*)
This is a GMM model of the input data. It is memory efficient so that it can process very large input array like objects.

The mixtures are incrementally added by splitting the heaviest component in two components and perturbation of the original mean.

expectation(*x*)
Evaluate one example

fit(*X*)

load_model(*file_name*)
Load the model from a pickle.load

log_multivariate_normal_density_diag(*x*, *means=0.0*, *covars=1.0*)
Compute Gaussian log-density at X for a diagonal model

mixup(*n_new_mixies*)
Add n new mixies to the mixture.

save_model(*file_name*)
Save the GMM model as a pickle.

score(*x*)
Get the log prob of the x variable being generated by the mixture.

alex.ml.lbp package

Submodules

alex.ml.lbp.node module

class alex.ml.lbp.node.**DiscreteFactor**(*name*, *desc*, *prob_table*)
Bases: *alex.ml.lbp.node.Factor*

This is a base class for discrete factor nodes in the Bayesian Network.

It can works with full conditional table defined by the provided prob_table function.

The variables must be attached in the same order as are the parameters in the prob_table function.

get_output_message(*variable*)
Returns output messages from this factor to the given variable node.

update_input_messages()
Updates all input messages from connected variable nodes.

class alex.ml.lbp.node.**DiscreteNode**(*name*, *desc*, *card*, *observed=False*)
Bases: *alex.ml.lbp.node.VariableNode*

This is a class for all nodes with discrete/enumerable values.

The probabilities are stored in log format.

copy_node(*node*)

explain(*full=False*, *linear_prob=False*)
This function prints the values and their probabilities for this node.

get_most_probable_value()
The function returns the most probable value and its probability in a tuple.

get_output_message(*factor*)
Returns output messages from this node to the given factor.

This is done by subtracting the input log message from the given factor node from the current estimate log probabilities in this node.

get_two_most_probable_values()
This function returns two most probable values and their probabilities.

The function returns a tuple consisting of two tuples (value, probability).

get_values()

normalise()
This function normalise the sum of all probabilities to 1.0

update_backward_messages()

update_forward_messages()

update_marginals()
Update the marginal probabilities in the node by collecting all input messages and summing them in the log domain.

Finally, probabilities are normalised to sum to 1.0.

class alex.ml.lbp.node.**Factor**(*name*, *desc*)
Bases: *alex.ml.lbp.node.GenericNode*

This is a base class for all factor nodes in the Bayesian Network.

attach_variable(*variable*)

detach_variable(*variable*)

get_variables()

class alex.ml.lbp.node.**GenericNode**(*name*, *desc*)
Bases: object

This is a base class for all nodes in the Bayesian Network.

class alex.ml.lbp.node.**VariableNode**(*name*, *desc*)
Bases: *alex.ml.lbp.node.GenericNode*

This is a base class for all variable nodes in the Bayesian Network.

attach_factor(*factor*, *forward=False*)

detach_factor(*factor*)

get_factors()

Module contents

Submodules

alex.ml.exceptions module

exception alex.ml.exceptions.**FFNNException**
Bases: *alex.AlexException*

exception alex.ml.exceptions.**NBListException**
Bases: *alex.AlexException*

alex.ml.features module

This module contains generic code for working with feature vectors (or, in general, collections of features).

class alex.ml.features.**Abstracted**
Bases: object

all_instantiations(*do_abstract=False*)

get_concrete()

get_generic()

instantiate(*type_*, *value*, *do_abstract=False*)

Example: Let self represent da1(a1=T1:v1)&da2(a2=T2:v2)&da3(a3=T1:v3).

Calling self.instantiate("T1", "v1") results in

da1(a1=T1)&da2(a2=v2)&da3(a3=v3) ..if do_abstract == False

da1(a1=T1)&da2(a2=v2)&da3(a3=T1_other) ..if do_abstract == True

Calling self.instantiate("T1", "x1") results in

da1(a1=x1)&da2(a2=v2)&da3(a3=v3) ..if do_abstract == False

da1(a1=T1_other)&da2(a2=v2)&da3(a3=T1_other) ..if do_abstract == True.

insts_for_type(*type_*)

insts_for_typeval(*type_*, *value*)

iter_instantiations()

iter_triples()

iter_typeval()
Iterates the abstracted items in self, yielding combined representations of the type and value of each such token. An abstract method of this class.

join_typeval(*type_*, *val*)

classmethod make_other(*type_*)

other_val = '[OTHER]'

replace_typeval(*combined*, *replacement*)

splitter = '='

to_other()

class alex.ml.features.**AbstractedTuple2**
Bases: alex.ml.features.AbstractedFeature

class alex.ml.features.**Features**(**args*, ***kwargs*)
Bases: object

A mostly abstract class representing features of an object.

Attributes: features: mapping of the features to their values set: set of the features

classmethod do_with_abstract(*feature*, *meth*, **args*, ***kwargs*)

get_feature_coords_vals(*feature_idxs*)
Builds the feature vector based on the provided mapping of features onto their indices. Returns the vector as a two lists, one of feature coordinates, one of feature values.

Arguments: feature_idxs: a mapping { feature : feature index }

get_feature_vector(*feature_idxs*)
Builds the feature vector based on the provided mapping of features onto their indices.

Arguments: feature_idxs: a mapping { feature : feature index }

classmethod iter_abstract(*feature*)

iter_instantiations()

iteritems()
Iterates tuples of this object's features and their values.

classmethod join(*feature_sets*, *distinguish=True*)
Joins a number of sets of features, keeping them distinct.

Arguments:

distinguish – whether to treat the feature sets as of different types (distinguish=True) or just merge features from them by adding their values (distinguish=False). Default is True.

Returns a new instance of JoinedFeatures.

prune(*to_remove=None*, *min_val=None*)
Discards specified features.

Arguments: to_remove – collection of features to be removed min_val – threshold for feature values in order for them to be

retained (those not meeting the threshold are pruned)

class alex.ml.features.**JoinedFeatures**(*feature_sets*)
Bases: *alex.ml.features.Features*

JoinedFeatures are indexed by tuples (feature_sets_index, feature) where feature_sets_index selects the required set of features. Sets of features are numbered with the same indices as they had in the list used to initialise JoinedFeatures.

Attributes: features: mapping { (feature_set_index, feature) : value of feature } set: set of the (feature_set_index, feature) tuples generic: mapping { (feature_set_index, abstracted_feature) :

generic_feature }

instantiable: mapping { feature [generic part of feature } for] features from self.features.keys() that are abstracted

iter_instantiations()

class alex.ml.features.**ReplaceableTuple2**
Bases: tuple

iter_combined()

replace(*old*, *new*)

to_other()

alex.ml.features.**make_abstract**(*replaceable*, *iter_meth=None*, *replace_meth=None*, *splitter='='*, *make_other=None*)

alex.ml.features.**make_abstracted_tuple**(*abstr_idxs*)
Example usage:

AbTuple2 = make_abstract_tuple((2,)) ab_feat = AbTuple2((dai.dat, dai.name,

'='.join(dai.name.upper(), dai.value)))

... ab_feat.instantiate('food', 'chinese') ab_feat.instantiate('food', 'indian')

alex.ml.ffnn module

class alex.ml.ffnn.**FFNN**
Bases: object

Implements simple feed-forward neural network with:

– input layer - activation function linear – hidden layers - activation function tanh – output layer - activation function softmax

add_layer(*w*, *b*)
Add next layer into the network.

Parameters

- **w** – next layer weights
- **b** – next layer biases

Returns none

load(*file_name*)
Loads saved NN.

Parameters **file_name** – file name of the saved NN

Returns None

predict(*input*)
Returns the output of the last layer.

As it is output of a layer with softmax activation function, the output is a vector of probabilities of the classes being predicted.

Parameters **input** – input vector for the first NN layer.

Returns return the output of the last activation layer

save(*file_name*)
Saves the NN into a file.

Parameters **file_name** – name of the file where the NN will be saved

Returns None

set_input_norm(*m*, *std*)

sigmoid(*y*)

softmax(*y*)

tanh(*y*)

alex.ml.hypothesis module

This module collects classes representing the uncertainty about the actual value of a base type instance.

class alex.ml.hypothesis.**ConfusionNetwork**
Bases: *alex.ml.hypothesis.Hypothesis*

Confusion network. In this representation, each fact breaks down into a sequence of elementary acts.

add(*probability*, *fact*)
Append a fact to the confusion network.

add_merge(*p*, *fact*, *combine=u'max'*)
Add a fact and if it exists merge it according to the given combine strategy.

extend(*conf_net*)

classmethod from_fact(*fact*)
Constructs a deterministic confusion network that asserts the given 'fact'. Note that 'fact' has to be an iterable of elementary acts.

get_prob(*fact*)
Get the probability of the fact.

merge(*conf_net*, *combine=u'max'*)
Merges facts in the current and the given confusion networks.

Arguments:

combine – can be one of {'new', 'max', 'add', 'arit', 'harm'}, and determines how two probabilities should be merged (default: 'max')

XXX As of now, we know that different values for the same slot are contradictory (and in general, the set of contradicting attr-value pairs could be larger). We should therefore consider them alternatives to each other.

normalise()
Makes sure that all probabilities add up to one. They should implicitly sum to one: p + (1-p) == 1.0

prune(*prune_prob=0.005*)
Prune all low probability dialogue act items.

remove(*fact_to_remove*)

sort(*reverse=True*)

update_prob(*probability*, *fact*)
Update the probability of a fact.

exception alex.ml.hypothesis.**ConfusionNetworkException**
Bases: exceptions.Exception

class alex.ml.hypothesis.**Hypothesis**
Bases: object

This is the base class for all forms of probabilistic hypotheses representations.

classmethod from_fact(*fact*)
Constructs a deterministic hypothesis that asserts the given 'fact'.

class alex.ml.hypothesis.**NBList**
Bases: *alex.ml.hypothesis.Hypothesis*

This class represents the uncertainty using an n-best list.

When updating an N-best list, one should do the following.

1. add utterances or parse a confusion network
2. merge and normalise, in either order

add(*probability*, *fact*)
Finds the last hypothesis with a lower probability and inserts the new item before that one. Optimised for adding objects from the highest probability ones to the lowest probability ones.

add_other(*other*)
The N-best list is extended to include the other object to represent those object values that are not enumerated in the list.

Returns self.

classmethod from_fact(*fact*)

get_best()
Returns the most probable value of the object.

merge()
Adds up probabilities for the same hypotheses. Returns self.

normalise()
Scales the list to sum to one.

alex.ml.logarithmetic module

alex.ml.logarithmetic.**add**(*a*, *b*)
Computes pairwise addition of two vectors in the log domain.

This is equivalent to [a1+b1, a2+b2, ...] in the linear domain..

alex.ml.logarithmetic.**devide**(*a*, *b*)
Computes pairwise division between vectors a and b in the log domain.

This is equivalent to [a1/b1, a2/b2, ...] in the linear domain.

alex.ml.logarithmetic.**dot**(*a*, *b*)
Computes dot product in the log domain.

This is equivalent to a1*b1+a2*b2+... in the linear domain.

alex.ml.logarithmetic.**linear_to_log**(*a*)
Converts a vector from the linear domain to the log domain.

alex.ml.logarithmetic.**log_to_linear**(*a*)
Converts a vector from the log domain to the linear domain.

alex.ml.logarithmetic.**multiply**(*a*, *b*)
Computes pairwise multiplication between vectors a and b in the log domain.

This is equivalent to [a1*b1, a2*b2, ...] in the linear domain.

alex.ml.logarithmetic.**normalise**(*a*)
normalises the input probability vector to sum to one in the log domain.

This is equivalent to a/sum(a) in the linear domain.

alex.ml.logarithmetic.**sub**(*a*, *b*)
Computes pairwise subtraction of two vectors in the log domain.

This is equivalent to [a1-b1, a2-b2, ...] in the linear domain.

alex.ml.logarithmetic.**sum**(*a*, *axis=None*)

alex.ml.test_hypothesis module

class alex.ml.test_hypothesis.**TestConfusionNetwork**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_iter()

test_remove()

alex.ml.tffnn module

Module contents

alex.tests package

Submodules

alex.tests.autopath module

self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing "pypy" is in sys.path.

If you modify the master "autopath.py" version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

alex.tests.test_asr_google module

alex.tests.test_mproc module

alex.tests.test_numpy_with_optimised_ATLAS module

alex.tests.test_numpy_with_optimised_ATLAS.**main**()

alex.tests.test_pyaudio module

alex.tests.test_tts_flite_en module

alex.tests.test_tts_google_cs module

alex.tests.test_tts_google_en module

alex.tests.test_tts_voice_rss_en module

Module contents

alex.tools package

Subpackages

alex.tools.mturk package

Subpackages

alex.tools.mturk.bin package

Submodules

alex.tools.mturk.bin.approve_all_HITs module

alex.tools.mturk.bin.autopath module self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing "pypy" is in sys.path.

If you modify the master "autopath.py" version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

alex.tools.mturk.bin.delete_all_HITs module

alex.tools.mturk.bin.delete_aproved_rejected_expired_HITs module

alex.tools.mturk.bin.expire_all_HITs module

alex.tools.mturk.bin.get_account_balance module

alex.tools.mturk.bin.mturk module

alex.tools.mturk.bin.mturk.**print_assignment**(*ass*)

alex.tools.mturk.bin.reject_HITs_from_worker module

Module contents

alex.tools.mturk.sds-evaluation package

Subpackages

alex.tools.mturk.sds-evaluation.common package

Submodules

alex.tools.mturk.sds-evaluation.common.lock_test module

alex.tools.mturk.sds-evaluation.common.mturk-ganalytics module

alex.tools.mturk.sds-evaluation.common.mturk-log module

alex.tools.mturk.sds-evaluation.common.mturk-logs-stats module

alex.tools.mturk.sds-evaluation.common.mturk-remote-addr module

alex.tools.mturk.sds-evaluation.common.utils module

Module contents

Submodules

alex.tools.mturk.sds-evaluation.autopath module

alex.tools.mturk.sds-evaluation.copy_feedbacks module

alex.tools.mturk.sds-evaluation.cued_feedback_stats module

alex.tools.mturk.sds-evaluation.cued_phone_number_stats module

Module contents

Module contents

alex.tools.vad package

Submodules

alex.tools.vad.autopath module self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing "pypy" is in sys.path.

If you modify the master "autopath.py" version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

alex.tools.vad.train_vad_gmm module

alex.tools.vad.train_vad_gmm.**load_mlf**(*train_data_sil_aligned*, *max_files*, *max_frames_per_segment*)

alex.tools.vad.train_vad_gmm.**mixup**(*gmm*, *vta*, *name*)

alex.tools.vad.train_vad_gmm.**train_gmm**(*name*, *vta*)

alex.tools.vad.train_vad_nn_theano module

Module contents

Submodules

alex.tools.apirequest module

class alex.tools.apirequest.**APIRequest**(*cfg*, *fname_prefix*, *log_elem_name*)

Bases: object

Handles functions related web API requests (logging).

class alex.tools.apirequest.**DummyLogger**(*stream=<open file '<stderr>', mode 'w'>*)

A dummy logger implementation for debugging purposes that will just print to STDERR or whatever output stream it is given in the constructor.

external_data_file(*dummy1*, *dummy2*, *data*)

get_session_dir_name()

info(*text*)

alex.tools.autopath module

self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing "pypy" is in sys.path.

If you modify the master "autopath.py" version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

Module contents

alex.utils package

Submodules

alex.utils.analytics module

alex.utils.audio module

alex.utils.audio_play module

alex.utils.autopath module

self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing "pypy" is in sys.path.

If you modify the master "autopath.py" version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

alex.utils.cache module

class alex.utils.cache.**Counter**
Bases: dict

Mapping where default values are zero

alex.utils.cache.**get_persitent_cache_content**(*key*)

alex.utils.cache.**lfu_cache**(*maxsize=100*)
Least-frequently-used cache decorator.

Arguments to the cached function must be hashable. Cache performance statistics stored in f.hits and f.misses. Clear the cache with f.clear(). http://en.wikipedia.org/wiki/Least_Frequently_Used

alex.utils.cache.**lru_cache**(*maxsize=100*)
Least-recently-used cache decorator.

Arguments to the cached function must be hashable. Cache performance statistics stored in f.hits and f.misses. Clear the cache with f.clear(). http://en.wikipedia.org/wiki/Cache_algorithms#Least_Recently_Used

alex.utils.cache.**persistent_cache**(*method=False*, *file_prefix=''*, *file_suffix=''*)
Persistent cache decorator.

It grows indefinitely. Arguments to the cached function must be hashable. Cache performance statistics stored in f.hits and f.misses.

alex.utils.cache.**set_persitent_cache_content**(*key*, *value*)

alex.utils.caminfodb module

class alex.utils.caminfodb.**CamInfoDb**(*db_path*)
Bases: object

get_by_id(*rec_id*)

get_matching(*query*)

get_possible_values()

get_slots()

matches(*rec*, *query*)

alex.utils.config module

class alex.utils.config.**Config**(*file_name=None*, *project_root=False*, *config={}*)
Bases: object

Config handles configuration data necessary for all the components in Alex. It is implemented using a dictionary so that any component can use arbitrarily structured configuration data.

Before the configuration file is loaded, it is transformed as follows:

1. **'{cfg_abs_path}' as a string anywhere in the file is replaced by an** absolute path of the configuration files. This can be used to make the configuration file independent of the location of programs that use it.

DEFAULT_CFG_PPATH = u'resources/default.cfg'

config_replace(*p*, *s*, *d=None*)
Replace a pattern p with string s in the whole config (recursively) or in a part of the config given in d.

contains(**path*)
Check if configuration contains given keys (= path in config tree).

get(*i*, *default=None*)

getpath(*path*, *default=None*)

load(*file_name*)

classmethod load_configs(*config_flist=[]*, *use_default=True*, *log=True*, **init_args*, ***init_kwargs*)
Loads and merges configs from paths listed in 'config_flist'. Use this method instead of direct loading configs, as it takes care of not only merging them but also processing some options in a special way.

Arguments:

config_flist – list of paths to config files to load and merge; order matters (default: [])

use_default – whether to insert the default config ($ALEX/resources/default.cfg) at the beginning of 'config_flist' (default: True)

log – whether to log the resulting config using the system logger (default: True)

init_args – additional positional arguments will be passed to constructors for each config

init_kwargs – additional keyword arguments will be passed to constructors for each config

load_includes()

merge(*other*)
Merges self's config with other's config and saves it as a new self's config.

Keyword arguments:

- **other: a Config object whose configuration dictionary to merge** into self's one

unfold_lists(*pattern*, *unfold_id_key=None*, *part=[]*)
Unfold lists under keys matching the given pattern into several config objects, each containing one item. If pattern is None, all lists are expanded.

Stores a string representation of the individual unfolded values under the unfold_id_key if this parameter is set.

Only expands a part of the whole config hash (given by list of keys forming a path to this part) if the path parameter is set.

update(*new_config*, *config_dict=None*)
Updates the nested configuration dictionary by another, potentially also nested dictionary.

Keyword arguments:

- new_config: the new dictionary to update with
- config_dict: the config dictionary to be updated

alex.utils.config.**as_project_path**(*path*)

alex.utils.config.**callback_download_progress**(*blocks*, *block_size*, *total_size*)
callback function for urlretrieve that is called when connection is created and when once for each block

Parameters

- **blocks** – number of blocks transferred so far
- **block_size** – in bytes

- **total_size** – in bytes, can be -1 if server doesn't return it

alex.utils.config.**is_update_server_reachble**()

alex.utils.config.**load_as_module**(*path*, *force=False*, *encoding=u'UTF-8'*, *text_transforms=[]*)
Loads a file pointed to by 'path' as a Python module with minimal impact on the global program environment. The file name should end in '.py'.

Arguments: path – path towards the file force – whether to load the file even if its name does not end in '.py'

encoding – character encoding of the file text_transforms – collection of functions to be run on the original file text

Returns the loaded module object.

alex.utils.config.**online_update**(*file_name*)
This function can download file from a default server if it is not available locally. The default server location can be changed in the config file.

The original file name is transformed into absolute name using as_project_path function.

Parameters **file_name** – the file name which should be downloaded from the server

Returns a file name of the local copy of the file downloaded from the server

alex.utils.config.**set_online_update_server**(*server_name*)
Set the name of the online update server. This function can be used to change the server name from inside a config file.

Parameters **server_name** – the HTTP(s) path to the server and a location where the desired data reside.

Returns None

alex.utils.config.**to_project_path**(*path*)
Converts a relative or absoulute file system path to a path relative to project root.

alex.utils.cuda module

alex.utils.cuda.**cudasolve**(*A*, *b*, *tol=0.001*, *normal=False*, *regA=1.0*, *regI=0.0*)
Conjugate gradient solver for dense system of linear equations.

Ax = b

Returns: x = A^(-1)b

If the system is normal, then it solves

(regA*A'A +regI*I)x= b

Returns: x = (A'A +reg*I)^(-1)b

alex.utils.czech_stemmer module

Czech stemmer Copyright © 2010 Luís Gomes <luismsgomes@gmail.com>.

Ported from the Java implementation available at: http://members.unine.ch/jacques.savoy/clef/index.html

alex.utils.czech_stemmer.**cz_stem**(*l*, *aggressive=False*)

alex.utils.czech_stemmer.**cz_stem_word**(*word*, *aggressive=False*)

alex.utils.enums module

alex.utils.enums.**enum**(**sequential*, ***named*)
Useful for creating enumerations.

e.g.: DialogueType = enum(deterministic=0, statistical=1, mix=2)

alex.utils.env module

alex.utils.env.**root**()
Finds the root of the project and return it as string.

The root is the directory named alex.

alex.utils.excepthook module

Depending on the hook_type, ExceptionHook class adds various hooks how to catch exceptions.

class alex.utils.excepthook.**ExceptionHook**(*hook_type*, *logger=None*)
Bases: object

Singleton objects for registering various hooks for sys.exepthook. For registering a hook, use set_hook.

apply()
The object can be used to store settings for excepthook. a = ExceptionHook('log') # now it logs b = ExceptionHook('ipdb') # now it uses ipdb a.apply() # now it logs again

logger = None

classmethod set_hook(*hook_type=None*, *logger=None*)
Choose an exception hook from predefined functions.

hook_type: specify the name of the hook method

alex.utils.excepthook.**hook_decorator**(*f*)
Print the caution message when the decorated function raises an error.

alex.utils.excepthook.**ipdb_hook**(**args*, ***kwargs*)

alex.utils.excepthook.**log_and_ipdb_hook**(**args*, ***kwargs*)

alex.utils.excepthook.**log_hook**(**args*, ***kwargs*)

alex.utils.exceptions module

exception alex.utils.exceptions.**ConfigException**
Bases: *alex.AlexException*

exception alex.utils.exceptions.**SessionClosedException**
Bases: *alex.AlexException*

exception alex.utils.exceptions.**SessionLoggerException**
Bases: *alex.AlexException*

alex.utils.exdec module

alex.utils.exdec.**catch_ioerror**(*user_function*, *msg=''*)

alex.utils.filelock module

Context manager for locking on a file. Obtained from

http://www.evanfosmark.com/2009/01 /cross-platform-file-locking-support-in-python/,

licensed under BSD.

This is thought to work safely on NFS too, in contrast to fcntl.flock(). This is also thought to work safely over SMB and else, in contrast to fcntl.lockf(). For both issues, consult http://oilq.org/fr/node/13344.

Use as simply as

with FileLock(filename): <critical section for working with the file at 'filename'>

class alex.utils.filelock.**FileLock**(*file_name*, *timeout=10*, *delay=0.05*)
Bases: object

A file locking mechanism that has context-manager support so you can use it in a with statement. This should be relatively portable as it doesn't rely on msvcrt or fcntl for the locking.

acquire()
Acquire the lock, if possible. If the lock is in use, it check again every 'wait' seconds. It does this until it either gets the lock or exceeds 'timeout' number of seconds, in which case it throws an exception.

release()
Get rid of the lock by deleting the lockfile. When working in a 'with' statement, this method gets automatically called at the end.

exception alex.utils.filelock.**FileLockException**
Bases: exceptions.Exception

alex.utils.fs module

Filesystem utility functions.

class alex.utils.fs.**GrepFilter**(*stdin*, *stdout*, *breakchar=u'n'*)
Bases: multiprocessing.process.Process

add_listener(*regex*, *callback*)
Adds a listener to the output strings.

Arguments:

regex – the compiled regular expression to look for ('regex.search') in any piece of output

callback – a callable that is invoked for output where 'regex' was found. This will be called like this:

outputting &= callback(output_unicode_str)

That means, callback should take the unicode string argument containing what would have been output and return a boolean value which is True iff outputting should stop.

Returns the index of the listener for later reference.

flush(*force=True*)

remove_listener(*listener_idx*)

run()

write(*unistr*)

alex.utils.fs.**find**(*dir_*, *glob_*, *mindepth=2*, *maxdepth=6*, *ignore_globs=[]*, *ignore_paths=None*, *follow_symlinks=True*, *prune=False*, *rx=None*, *notrx=None*)

A simplified version of the GNU 'find' utility. Lists files with basename matching '**glob_**' found in '**dir_**' in depth between 'mindepth' and 'maxdepth'.

The 'ignore_globs' argument specifies a glob for basenames of files to be ignored. The 'ignore_paths' argument specifies a collection of real absolute pathnames that are pruned from the search. For efficiency reasons, it should be a set.

In the current implementation, the traversal resolves symlinks before the file name is checked. However, taking symlinks into account can be forbidden altogether by specifying 'follow_symlinks=False'. Cycles during the traversal are avoided.

- prune: whether to prune the subtree below a matching directory
- **rx: regexp to use as an additional matching criterion apart from** '**glob_**'; the 're.match' function is used, as opposed to 're.find'
- notrx: like 'rx' but this specifies the regexp that must NOT match

The returned set of files consists of real absolute pathnames of those files.

alex.utils.fs.**normalise_path**(*path*)

Normalises a filesystem path using tilde expansion, absolutising and normalising the path, and resolving symlinks.

alex.utils.fs.**test_grep_filter**()

alex.utils.htk module

class alex.utils.htk.**Features**(*file_name=None*)

Read HTK format feature files

open(*file_name*)

class alex.utils.htk.**MLF**(*file_name=None*, *max_files=None*)

Read HTK MLF files.

Def: segment is a sequence of frames with the same label.

count_length(*pattern*)

Count length of all segments matching the pattern

filter_zero_segments()

Remove aligned segments which have zero length.

merge()

Merge the consecutive segments with the same label into one segment.

open(*file_name*)

shorten_segments(*n=100*)

Shorten segments to n-frames.

sub(*pattern*, *repl*, *pos=True*)

times_to_frames(*frame_length=0.01*)

times_to_seconds()

trim_segments(*n=3*)

Remove n-frames from the beginning and the end of a segment.

class alex.utils.htk.**MLFFeaturesAlignedArray**(*filter=None*)
Creates array like object from multiple mlf files and corresponding audio data. For each aligned frame it returns a feature vector and its label.

If a filter is set to a particular value, then only frames with the label equal to the filer will be returned. In this case, the label is not returned when iterating through the array.

append_mlf(*mlf*)
Add a mlf file with aligned transcriptions.

append_trn(*trn*)
Adds files with audio data (param files) based on the provided pattern.

get_frame(*file_name*, *frame_id*)
Returns a frame from a specific param file.

get_param_file_name(**args*, ***kwds*)
Returns the matching param file name.

class alex.utils.htk.**MLFMFCCOnlineAlignedArray**(*windowsize=250000*, *targetrate=100000*, *filter=None*, *usec0=False*, *usedelta=True*, *useacc=True*, *n_last_frames=0*, *mel_banks_only=False*)
Bases: *alex.utils.htk.MLFFeaturesAlignedArray*

This is an extension of MLFFeaturesAlignedArray which computes the features on the fly from the input wav files.

It uses our own implementation of the MFCC computation. As a result it does not give the same results as the HTK HCopy.

The experience suggests that our MFFC features are worse than the features generated by HCopy.

get_frame(*file_name*, *frame_id*)
Returns a frame from a specific param file.

alex.utils.interface module

class alex.utils.interface.**Interface**
Bases: object

alex.utils.interface.**interface_method**(*f*)

alex.utils.lattice module

alex.utils.mfcc module

class alex.utils.mfcc.**MFCCFrontEnd**(*sourcerate=16000*, *framesize=512*, *usehamming=True*, *preemcoef=0.97*, *numchans=26*, *ceplifter=22*, *numceps=12*, *enormalise=True*, *zmeansource=True*, *usepower=True*, *usec0=True*, *usecmn=False*, *usedelta=True*, *useacc=True*, *n_last_frames=0*, *lofreq=125*, *hifreq=3800*, *mel_banks_only=False*)
This is an a CLOSE approximation of MFCC coefficients computed by the HTK.

The frame size should be a number of power of 2.

TODO: CMN is not implemented. It should normalise only teh cepstrum, not the delta or acc coefficients.

It was not tested to give exactly the same results the HTK. As a result, it should not be used in conjunction with models trained on speech parametrised with the HTK.

Over all it appears that this implementation of MFCC is worse than the one from the HTK. On the VAD task, the HTK features score 90.8% and the this features scores only 88.7%.

freq_to_mel(*freq*)

init_cep_liftering_weights()

init_hamming()

init_mel_filter_bank()
Initialise the triangular mel freq filters.

mel_to_freq(*mel*)

param(*frame*)
Compute the MFCC coefficients in a way similar to the HTK.

preemphasis(*frame*)

class alex.utils.mfcc.**MFCCKaldi**(*sourcerate=16000*, *framesize=512*, *usehamming=True*, *preemcoef=0.97*, *numchans=26*, *ceplifter=22*, *numceps=12*, *enormalise=True*, *zmeansource=True*, *usepower=True*, *usec0=True*, *usecmn=False*, *usedelta=True*, *useacc=True*, *n_last_frames=0*, *lofreq=125*, *hifreq=3800*, *mel_banks_only=False*)
TODO port Kaldi mfcc to Python. Use similar parameters as in suggested in __init__ function

param(*frame*)
Compute the MFCC coefficients in a way similar to the HTK.

alex.utils.mproc module

Implements useful classes for handling multiprocessing implementation of the Alex system.

class alex.utils.mproc.**InstanceID**
Bases: object

This class provides unique ids to all instances of objects inheriting from this class.

get_instance_id(**args*, ***kw*)

instance_id = <Synchronized wrapper for c_int(0)>

lock = <Lock(owner=None)>

class alex.utils.mproc.**SystemLogger**(*output_dir*, *stdout_log_level='DEBUG'*, *stdout=True*, *file_log_level='DEBUG'*)
Bases: object

This is a multiprocessing-safe logger. It should be used by all components in Alex.

critical(**args*, ***kwargs*)

debug(**args*, ***kwargs*)

error(**args*, ***kwargs*)

exception(*message*)

formatter(**args*, ***kw*)
Format the message - pretty print

get_session_dir_name(**args*, ***kw*)
Return directory where all the call related files should be stored.

get_time_str()
Return current time in dashed ISO-like format.

It is useful in constructing file and directory names.

info(**args*, ***kwargs*)

levels = {'INFO': 20, 'CRITICAL': 40, 'EXCEPTION': 50, 'SYSTEM-LOG': 0, 'WARNING': 30, 'ERROR': 60, 'DEB

lock = <RLock(None, 0)>

log(**args*, ***kw*)
Logs the message based on its level and the logging setting. Before writing into a logging file, it locks the file.

session_end(**args*, ***kw*)
WARNING: Deprecated Disables logging into the session-specific directory.

We better do not end a session because very often after the session_end() method is called there are still incoming messages. Therefore, it is better to wait for the session_start() method to set a new destination for the session log.

session_start(**args*, ***kw*)
Create a specific directory for logging a specific call.

NOTE: This is not completely safe. It can be called from several processes.

session_system_log(**args*, ***kwargs*)
This logs specifically only into the call-specific system log.

warning(**args*, ***kwargs*)

alex.utils.mproc.**async**(*func*)
A function decorator intended to make "func" run in a separate thread (asynchronously). Returns the created Thread object

E.g.: @async def task1():

do_something

@async def task2():

do_something_too

t1 = task1() t2 = task2() ... t1.join() t2.join()

alex.utils.mproc.**etime**(*name='Time'*, *min_t=0.3*)
This decorator measures the execution time of the decorated function.

alex.utils.mproc.**file_lock**(*file_name*)
Multiprocessing lock using files. Lock on a specific file.

alex.utils.mproc.**file_unlock**(*lock_file*)
Multiprocessing lock using files. Unlock on a specific file.

alex.utils.mproc.**global_lock**(*lock*)
This decorator makes the decorated function thread safe.

Keyword arguments: lock – a global variable pointing to the object to lock on

alex.utils.mproc.**local_lock**()
This decorator makes the decorated function thread safe.

For each function it creates a unique lock.

alex.utils.nose_plugins module

alex.utils.parsers module

class alex.utils.parsers.**CamTxtParser**(*lower=False*)
Bases: object

Parser of files of the following format: <<BOF>> [record]

[record]

... <<EOF>>

where [record] has the following format:

<<[record]>> [property name]([property value]) <</[record]>>

[property name] and [property value] are arbitrary strings

Any ” or ‘ characters are stripped from the beginning and end of each [property value].

line_expr = <_sre.SRE_Pattern object>

parse(*f_obj*)
Parse the given file and return list of dictionaries with parsed values.

Arguments: f_obj – filename of file or file object to be parsed

alex.utils.procname module

alex.utils.procname.**get_proc_name**()

alex.utils.procname.**set_proc_name**(*newname*)

alex.utils.rdb module

class alex.utils.rdb.**Rdb**(*port=4446*)
Bases: pdb.Pdb

do_c(*arg*)

do_cont(*arg*)

do_continue(*arg*)

alex.utils.sessionlogger module

class alex.utils.sessionlogger.**SessionLogger**
Bases: multiprocessing.process.Process

This is a multiprocessing-safe logger. It should be used by Alex to log information according the SDC 2010 XML format.

Date and times should also include time zone.

Times should be in seconds from the beginning of the dialogue.

cancel_join_thread()

run()

set_cfg(*cfg*)

set_close_event(*close_event*)

alex.utils.test_analytics module

alex.utils.test_fs module

Unit tests for alex.util.fs.

class alex.utils.test_fs.**TestFind**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()
Creates a playground of a directory tree. It looks like this: <testroot>/

- **a/**
  - aa/
  - ab/
  - **ac/**
    - **aca/**
      - acaa/
      - acab -> baaaa
- **b/**
  - **ba/**
    - **baa/**
      - **baaa/**
        baaaa -> daaa
        baaab/
        baaaba/
        baaabaa/
        baaabab -> ca
- **c/**
  - **ca/**
    - caa/
    - **cab/**
      - caba/
    - cac -> db
- **d/**
  - **da/**

        * **daa/**
            - daaa -> acab
    - db -> baaaba

tearDown()
Deletes the mock-up directory tree.

test_cycles()
Test the processing of cycles in the directory structure.

test_depth()
Tests mindepth and maxdepth.

test_globs()
Tests processing of the selection glob.

test_ignore_globs()
Test the functionality of ignore globs.

test_symlinks1()
Basic test for symlinks.

test_wrong_args()
Test for handling wrong arguments.

alex.utils.test_sessionlogger module

class alex.utils.test_sessionlogger.**TestSessionLogger**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_session_logger()

alex.utils.test_text module

class alex.utils.test_text.**TestString**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_parse_command()

test_split_by()

alex.utils.text module

class alex.utils.text.**Escaper**(*chars=u'"'*, *escaper=u'\'*, *re_flags=0*)
Bases: object

Creates a customised escaper for strings. The characters that need escaping, as well as the one used for escaping can be specified.

ESCAPED = 1

ESCAPER = 0

NORMAL = 2

annotate(*esced*)
Annotates each character of a text that has been escaped whether:

Escaper.ESCAPER - it is the escape character Escaper.ESCAPED - it is a character that was escaped Escaper.NORMAL - otherwise.

It is expected that only parts of the text may have actually been escaped.

Returns a list with the annotation values, co-indexed with characters of the input text.

escape(*text*)
Escapes the text using the parameters defined in the constructor.

static re_literal(*char*)
Escapes the character so that when it is used in a regexp, it matches itself.

static re_literal_list(*chars*)
Builds a [] group for a regular expression that matches exactly the characters specified.

unescape(*text*)
Unescapes the text using the parameters defined in the constructor.

alex.utils.text.**escape_special_characters_shell**(*text*, *characters=u'\"'*)
Simple function that tries to escape quotes. Not guaranteed to produce the correct result!! If that is needed, use the new 'Escaper' class.

alex.utils.text.**findall**(*text*, *char*, *start=0*, *end=-1*)

alex.utils.text.**min_edit_dist**(*target*, *source*)
Computes the min edit distance from target to source.

alex.utils.text.**min_edit_ops**(*target*, *source*, *cost=<function <lambda>>*)
Computes the min edit operations from target to source.

Parameters
- **target** – a target sequence
- **source** – a source sequence
- **cost** – an expression for computing cost of the edit operations

Returns a tuple of (insertions, deletions, substitutions)

alex.utils.text.**parse_command**(*command*)
Parse the command name(var1="val1",...) into a dictionary structure:

E.g. call(destination="1245",opt="X") will be parsed into:
{ "__name__": "call", "destination": "1245", "opt": "X"}
Return the parsed command in a dictionary.

alex.utils.text.**split_by**(*text*, *splitter*, *opening_parentheses=u''*, *closing_parentheses=u''*, *quotes=u'\"'*)
Splits the input text at each occurrence of the splitter only if it is not enclosed in parentheses.

text - the input text string splitter - multi-character string which is used to determine the position of splitting of the text

opening_parentheses - an iterable of opening parentheses that has to be respected when splitting, e.g. "{(" (default: '')

closing_parentheses - an iterable of closing parentheses that has to be respected when splitting, e.g. "})" (default: '')

quotes - an iterable of quotes that have to come in pairs, e.g. '"'

alex.utils.text.**split_by_comma**(*text*)

alex.utils.token module

alex.utils.token.**get_token**(*cfg*)

alex.utils.ui module

alex.utils.ui.**getTerminalSize**()
Retrieves the size of the current terminal window.

Returns (None, None) in case of lack of success.

alex.utils.various module

alex.utils.various.**crop_to_finite**(*val*)

alex.utils.various.**flatten**(*list_*, *ltypes=(<type 'list'>, <type 'tuple'>)*)
Flatten nested list into a simple list.

alex.utils.various.**get_text_from_xml_node**(*node*)
Get text from all child nodes and concatenate it.

alex.utils.various.**group_by**(*objects*, *attrs*)
Groups 'objects' by the values of their attributes 'attrs'.

Returns a dictionary mapping from a tuple of attribute values to a list of objects with those attribute values.

class alex.utils.various.**nesteddict**
Bases: collections.defaultdict

walk()

alex.utils.various.**remove_dups_stable**(*l*)
Remove duplicates from a list but keep the ordering.

@return: Iterator over unique values in the list

alex.utils.various.**split_to_bins**(*A*, *S=4*)
Split the A array into bins of size N.

Module contents

class alex.utils.**DummyLogger**
Bases: object

alex.utils.**one**()

alex.utils.**script_path**(*fname*, **args*)
Return path relative to the directory of the given file, and join the additional path parts.
Args: fname (str): file used to determine the root directory args (list): additional path parts

2.1.2 Submodules

2.1.3 alex.autopath module

self cloning, automatic path configuration

copy this into any subdirectory of pypy from which scripts need to be run, typically all of the test subdirs. The idea is that any such script simply issues

import autopath

and this will make sure that the parent directory containing "pypy" is in sys.path.

If you modify the master "autopath.py" version (in pypy/tool/autopath.py) you can directly run it which will copy itself on all autopath.py files it finds under the pypy root directory.

This module always provides these attributes:

pypydir pypy root directory path this_dir directory where this autopath.py resides

2.1.4 Module contents

exception `alex.`**`AlexException`**
Bases: `exceptions.Exception`

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

CHAPTER 4

How to write documentation

- Quick cheatsheet for ReST and Sphinx
- More thorough documentation with code examples

[HTKBook] The HTK Book, version 3.4

a

d

A

B

C

D

E

G

H

I

J

K

L

M

N

O

P

Q

R

T

U

V

W

Y

Z