PyDic Documentation

Release 1.1

Krzysztof Dorosz

May 31, 2014

Contents

Contents:

CHAPTER 1

Introduction

PyDic is a set of shell tools (being also python modules) for managing simple inflectional dictionaries. It uses own dictionary format based on Berkeley DB and Marisa Trie data structures. Dictionaries can be easily build with provided tools from raw UTF-8 text files in simple format.

PyDic supports using multiple different dictionaries at the same time.

1.1 Preparing a PyDic dictionary

We will focus on example Polish inflectional dictionary. There is an open dictionary called SJP that can be downloaded and used on variety of licenses (currently GPL, LGPL, CC SA).

Download and unzip SJP dictionary. This will usually produce the file *odm.txt*":

```
$ wget http://sjp.pl/slownik/odmiany/sjp-odm-20130802.zip
$ unzip sjp-odm-20130802.zip
$ ls
README.txt           odm.txt              sjp-odm-20130802.zip
```

Next step is to convert this file into a format that is PyDic compatible text input (which in that case is just converting file to UTF-8 encoding). For that there is provided very simple shell script:

```
$ sjp2pydic odm.txt > sjp.txt
```

Everything is now ready to create a pydic dictionary for sjp:

```
$ pydic_create.py -f sjp.txt
```

This command will create a dictionary represented by *sjp.pydic* directory.

1.2 Quickstart

To test working of simple SJP dictionary just try:

```
>>> from pydic import PyDic
>>> dic = PyDic('sjp.pydic')
>>> dic.id(u'komputerowi')
[PyDicId(u'67744@sjp'), PyDicId(u'67748@sjp')]
>>> dic.id_forms('67744@sjp')
[u'komputer', u'komputera', u'komputerach', u'komputerami', u'komputerem', u'komputerom', u'komputero
```

CHAPTER 2

Preparing dictionaries

To speed things up every dictionary you are going to use should be indexed first. You can easily create one using provided unix tool named `pydic_create.py`. After creating and index from flat text file technically you don't need it any more.

2.1 Syntax

```
$ pydic_create.py --help
usage: pydic_create.py [-h] [-d DELIMITER] [-f DICTIONARY_FILE] [-t TARGET]
                       [-n NAME] [-v]

Generate dictionary from text input.

optional arguments:
  -h, --help            show this help message and exit
  -d DELIMITER, --delimiter DELIMITER
                        delimiter
  -f DICTIONARY_FILE, --dictionary-file DICTIONARY_FILE
                        path to file with text dictionary
  -t TARGET, --target TARGET
                        path to target dictionary directory
  -n NAME, --name NAME  name of newly created dictionary (default same as text
                        file)
  -v, --verbose
```

2.2 Input format

Program takes as an input a file (or if not given with `-f` option `stdin` will be used) with plain text format `UTF-8` dictionary description.

Format of input is very simple and assumes that in each line there is one word with full inflectional vector:

```
<base form | form 1>:<form 2>:<form 3>: ... :<form n>
```

Warning: `<form 1>` always refers as `<base form>` as this is the first form in inflectional vector.

You can omit giving dictionary name if you use `-f` argument. Name will be taken by default from dictionary filename (extracting `.txt` or `.text` extension). When you do not provide `-f` argument and feed data from `stdin` providing

at least `-n` argument is obligatory.

2.3 Output

Program produces files into specified by argument `-t` (`--target`) directory. The files are being generated are:

- `.pydic` – this file is a flag that this is a PyDic dictionary folder. At the same time it stores dictionary name which then is used to identify dictionary and as a part of global word identificator used by PyDicManager.
- `forms.hash` – this file stores DB Berkeley Hash, which have mappings `"word form" -> "<id 1>:<id 2>:...<id 3>"`
- `forms.recno` – this file stores DB Berkeley Recno, which stores following mapping: `id -> "<prefix>:<suffix of form 1>:<suffix of form 2>: ... :<suffix of form n>"`

Note: All forms that are used as hash keys (index search) are lowercased before saved. Forms that are saved in recno are in original case. This means that if you provide form vector like `IBM:*:IBM:IBMu:IBMowi` you will get this vector both queering forms like *ibmowi* and *IBMOWI*.

Note: If argument `-t` (`--target`) is omitted current directory will be used.

Warning: You are supposed never have to touch this internal data structures. This information is given only for your convenience.

2.4 Afterwords

Target directory will be created with all required subdirectories if needed. Program will refuse to create dictionary if in specified target directory there are any of three mentioned above files:

```
$ pydic_create.py -f sjp.txt  -t sjp
Generating sjp dictionary
!!!Configuration Error: Cowardly refusing to create dictionary in non empty directory
```

You can move dictionary directory freely and also you can edit name of dictionary in `.pydic` file if needed (for example to create a branch). Because you will use this dictionary always referring to its filesystem location you need to only remember path to the dictionary.

2.5 Example

Here is an example of creating very small (only 16 words) dictionary:

```
$ cat sjp.10.txt
ablaktacja, ablaktacjach, ablaktacjami, ablaktacją, ablaktacje, ablaktację, ablaktacji, ablaktacjo, a
ablaktowalny, ablaktowalna, ablaktowalną, ablaktowalne, ablaktowalnego, ablaktowalnej, ablaktowalnemu
ablativus, ablativach, ablativami, ablativem, ablativie, ablativom, ablativowi, ablativów, ablativu,
ablatiwus, ablatiwach, ablatiwami, ablatiwem, ablatiwie, ablatiwom, ablatiwowi, ablatiwów, ablatiwu,
ablatyw, ablatywach, ablatywami, ablatywem, ablatywie, ablatywom, ablatywowi, ablatywów, ablatywu, ab
ablatywny, ablatywna, ablatywną, ablatywne, ablatywnego, ablatywnej, ablatywnemu, ablatywni, ablatywn
```

```
ablegat, ablegaci, ablegacie, ablegata, ablegatach, ablegatami, ablegatem, ablegatom, ablegatowi, abl
ablegier, ablegra, ablegrach, ablegrami, ablegrem, ablegrom, ablegrowi, ablegrów, ablegry, ablegrze
ablucja, ablucjach, ablucjami, ablucją, ablucje, ablucję, ablucji, ablucjo, ablucjom, ablucyj
ablutomania, ablutomaniach, ablutomaniami, ablutomanią, ablutomanie, ablutomanię, ablutomanii, ablut

$ pydic_create.py -f sjp.10.txt -v
Generating sjp.10 dictionary in folder sjp.10.pydic
[ 1 ] ablaktacja
[ 2 ] ablaktowalny
[ 3 ] ablativus
[ 4 ] ablatiwus
[ 5 ] ablatyw
[ 6 ] ablatywny
[ 7 ] ablegat
[ 8 ] ablegier
[ 9 ] ablucja
[ 10 ] ablutomania
```

You can also run this command using `stdin`:

```
$ cat sjp.10.txt | pydic_create.py -v
Generating sjp.10 dictionary in folder sjp.10.pydic
[ 1 ] ablaktacja
[ 2 ] ablaktowalny
[ 3 ] ablativus
[ 4 ] ablatiwus
[ 5 ] ablatyw
[ 6 ] ablatywny
[ 7 ] ablegat
[ 8 ] ablegier
[ 9 ] ablucja
[ 10 ] ablutomania
```

Warning: Remember to use `-n` option to give a name for a dictionary when using `stdin` input.

CHAPTER 3

PyDic - Single dictionary API

PyDic class is the most low-level API access to the dictionaries. It is responsible for handling single dictionary.

class pydic.**PyDic**(*path*)
Abstraction layer for accessing single dictionary

3.1 Loading dictionary

PyDic supports both formats. Pre-indexed format of *.pydic* directories which can be generated using *pydic_create.py* shell script or just plain text file. It automatically detects which one was used:

```
>>> from pydic import PyDic
>>> PyDic('sjp.pydic/') # pre-indexed pydic
>>> PyDic('sjp.txt')    # flat text dictionary
```

Note: Using pre-indexed PyDic is highly recommended due to indexing overhead. Loading plain text is mainly provided for tests and debug needs.

3.2 Word identification namespace

PyDic internally uses identifiers which are of *PyDicId* type, internally numbered from 1 to total number of words in the dictionary. PyDicId allows you to exactly identify a given inflectional vector. However dictionary can also be queried in simplified way, that is without using PyDicId. There is a method naming convention that can help you distinguish which methods are ID-based and which one will simply take any word form (unicode) as an argument. All methods starting with `word_` prefix need as an argument a unicode string (a word) and will return also list of unicodes. Other methods starts with `id_` prefix means that you will query dictionary using PyDicId.

Note: All methods starting with `word_` prefix will return a list in first place. This is because, there can be more than one inflectional vector that can be matched to a given as an argument word form.

Warning: Every single PyDic dictionary uses its own identifiers namespace numbered from 1 with concatenated dictionary name to it separated by @ sign. This helps to avoid identifies name clashing if you want to use more than one dictionary in your program. Consider also using PyDicManager for managing multiple dictionaries at the same time

Example:

```
from pydic import PyDic
dic = PyDic('sjp.pydic')
```

3.3 id method

PyDic.**id**(*form*)

Returns a list of PyDicId that match a given word form

Parameters **form** (*unicode*) – word form

Returns list of PyDicId or empty list

Example:

```
>>> dic.id(u'zamkowi')
[PyDicId(u'187274@sjp'), PyDicId(u'187275@sjp'), PyDicId(u'187358@sjp')]

>>> dic.id(u'zamek')
[PyDicId(u'187274@sjp'), PyDicId(u'187275@sjp')]
```

3.4 id_forms method

PyDic.**id_forms**(*pydic_id*)

Returns list of forms for a given PyDicId

Parameters **pydic_id** (*PyDicId, string*) – PyDicId or string id

Returns list of unicode strings or empty list

Example:

```
>>> dic.id_forms(PyDicId('187274@sjp'))
[u'zamek', u'zamka', u'zamkach', u'zamkami', u'zamki', u'zamkiem', u'zamkom', u'zamkowi', u'zamk\xf3w

>>> dic.id_forms(u'187274@sjp')
[u'zamek', u'zamka', u'zamkach', u'zamkami', u'zamki', u'zamkiem', u'zamkom', u'zamkowi', u'zamk\xf3w
```

3.5 word_forms method

PyDic.**word_forms**(*form*)

Returns list of list of forms for a given form

Parameters **form** (*unicode*) – word form

Returns list of lists of unicode strings or empty list

Example:

```
>>> dic.word_forms(u'zamek')
[[u'zamek', u'zamka', u'zamkach', u'zamkami', u'zamki', u'zamkiem', u'zamkom', u'zamkowi', u'zamk\xf3
```

Warning: As you can see there can be more than one inflectional vector that matches a given word. Therefore this function always return list of lists.

Warning: All forms that are used as hash keys are lowercased before used. Forms that are saved in recno are in original case. This means that if you provide form vector like `IBM:*:IBM:IBMu:IBMowi` you need to make all queries using lowered case forms like `ibmowi`, but as a result you will get correct vector form `[u'IBM', u'IBM', u'IBMowi']`. Quering for `IBMowi` will return empy result `[]`.

3.6 `id_base` method

`PyDic.`**`id_base`**(*pydic_id*)

Returns a base form of word given as PyDicId

Parameters **pydic_id** (*PyDicId, string*) – PyDicId

Returns unicode string or `None`

Example:

```
>>> dic.id_base('187274@sjp')
u'zamek'
>>> dic.id_base(PyDicId(u'187274@sjp'))
u'zamek'
```

3.7 `word_base` method

`PyDic.`**`word_base`**(*form*)

Returns a list of base forms of form

Parameters **form** (*unicode string*) – word form

Returns list of unicode strings or empty list

Example:

```
>>> dic.word_base(u'zamkowi')
[u'zamek', u'zamkowy']
```

Warning: As you can see there can be more than one inflectional vector that matches a given word. Therefore this function always return list of lists.

Note: Elements on that list are unique, it means that even if there are many lexems found but having the same base form, method will return 1-element list.

3.8 `a_id` method

`PyDic.`**`a_id`**(*form*)

Accents agnostic version of method `id()`

Parameters **form** (*unicode*) – form

Returns list of PyDicId or empty list

Example:

```
>>> dic.id(u'pszczoly')
[]
>>> dic.a_id(u'pszczoly')
[PyDicId(u'134051@sjp'), PyDicId(u'134056@sjp'), PyDicId(u'134050@sjp')]
```

Note: Default mapping is defined in `pydic.accents.AccentsTable.PL`

Warning: This method works using `pydic.Accents` class which is capable of generating a list of all possible word variants using given mapping from standard character to list of accent characters. Be warned that list of possibilities can be very long as this is number of combinations from all replaceable characters.

3.9 `a_word_forms` method

`PyDic.`**`a_word_forms`**(*form, mapping={u'a': [u'u0105'], u'A': [u'u0104'], u'c': [u'u0107'], u'Z': [u'u017b', u'u0179'], u'e': [u'u0119'], u'C': [u'u0106'], u'l': [u'u0142'], u'o': [u'xf3'], u'n': [u'u0144'], u's': [u'u015b'], u'N': [u'u0143'], u'O': [u'xd3'], u'E': [u'u0118'], u'z': [u'u017c', u'u017a'], u'S': [u'u015a'], u'L': [u'u0141']}*)

Accent agnostic version of word_forms method.

Parameters **form** (*unicode*) – word form

Returns list of lists of unicode strings or empty list

Example:

```
>>> dic.word_forms(u'pszczoly')
[]

>>> dic.a_word_forms(u'pszczoly')
[[u'pszczo\u0142a', u'pszczole', u'pszczo\u0142ach', u'pszczo\u0142ami', u'pszczo\u0142\u0105', u'psz
```

Note: Default mapping is defined in `pydic.accents.AccentsTable.PL`

Warning: This method works using `pydic.Accents` class which is capable of generating a list of all possible word variants using given mapping from standard character to list of accent characters. Be warned that list of possibilities can be very long as this is number of combinations from all replaceable characters.

3.10 `a_word_base` method

`PyDic.`**`a_word_base`**(*form*)

Accents agnostic version of `word_base()` method

Parameters **form** (*unicode string*) – word form

Returns list of unicode strings or empty list

Example:

```
>>> dic.word_base(u'pszczoly')
[]
```

```
>>> dic.a_word_base(u'pszczoly')
[u'pszczo\u0142a', u'pszczo\u0142y', u'Pszczo\u0142a']
```

Note: Default mapping is defined in `pydic.accents.AccentsTable.PL`

Warning: This method works using `pydic.Accents` class which is capable of generating a list of all possible word variants using given mapping from standard character to list of accent characters. Be warned that list of possibilities can be very long as this is number of combinations from all replaceable characters.

CHAPTER 4

PyDicManager - Multiple dictionary API

PyDicManager is a class that provides single and clean way to manage multiple dictionaries at the same time. Constructor method requires paths of all dictionaries that will be used, but after initialisation dictionaries will be referred by theirs names rather then paths. In fact, dictionary location (path) only matters on loading stage. Therefore, you can easily move your dictionaries to different place in filesystem as far as you only remember to point correct path when loading dictionary. Without changing an internal name of a dictionary (which is stored in file `.pydoc`, not in a name of a directory) no references of word identificators will be broken.

class `pydic.`**`PyDicManager`**`(`**args*`)`
Manages single access to multiple PyDic instances

As arguments simple define dictionary paths you want to load.

4.1 Word identification namespace

Because PyDic itself provides an word identification namespace, it does not make a big problem to handle multiple dictionaries at once.

Format of identificator is: `<pydic integer identificator>@<pydic name>`

Eg. `132@first_dictionary`

Warning: You should never load dictionaries with the same names, as dictionary name should be unique.

4.2 `id` method

`PyDicManager.`**`id`**`(`*word*`)`
Returns all known id for a word from every dictionary

Parameters **word** (*unicode*) – a word form

Returns list of str full id

Example:

```
>>>> dic.id(u'zamkowi')
[PyDicId('123643@gen12'), PyDicId('123644@gen12'), PyDicId('123802@gen12')]

>>>> dic.id(u'zamek')
[PyDicId('123643@gen12'), PyDicId('123644@gen12')]
```

Warning: Querying is case-insensitive.

4.3 `id_forms` method

PyDicManager.**id_forms**(*pydic_id*)

Returns forms vector for a full_id

Parameters **pydic_id** (*str*) – word full id

Returns list of unicode forms or empty list

Example:

```
>>> dic.id_forms('123643@gen12')
[u'zamek',
 u'zamka',
 u'zamkowi',
 u'zamek',
 u'zamkiem',
 u'zamku',
 u'zamku',
 u'zamki',
 u'zamk\xf3w',
 u'zamkom',
 u'zamki',
 u'zamkami',
 u'zamkach',
 u'zamki']
```

4.4 `word_forms` method

PyDicManager.**word_forms**(*word*)

For a word form returns a list of unique forms vector.

Parameters **word** (*unicode*) – a word form

Returns unique list of vector forms as tuple

Example:

```
>>> dic.word_forms(u'zamek')

[[u'zamek',
  u'zamka',
  u'zamkowi',
  u'zamek',
  u'zamkiem',
  u'zamku',
  u'zamku',
  u'zamki',
  u'zamk\xf3w',
  u'zamkom',
  u'zamki',
  u'zamkami',
  u'zamkach',
```

```
  u'zamki'],
 [u'zamek',
  u'zamek',
  u'zamku',
  u'zamkowi',
  u'zamek',
  u'zamkiem',
  u'zamku',
  u'zamku',
  u'zamki',
  u'zamk\xf3w',
  u'zamkom',
  u'zamki',
  u'zamkami',
  u'zamkach',
  u'zamki']]
```

Warning: Querying is case-insensitive.

Note: It is not possible to say which inflectional vector comes from which dictionary, as a returned list is flat. If you need this kind of information you will need make query by identificators. This method assumes that you want to be dictionary agnostic if querying by word forms, not by id.

Warning: As you can see there can be more than one inflectional vector that matches a given word. Therefore this function always return list of lists. PyDicManager will merge and will make unique all possible vectors from all possible dictionaries.

4.5 `id_base` method

`PyDicManager.`**`id_base`**(*pydic_id*)

Returns base form for a full id

Parameters **pydic_id** (*str*) – word full id

Returns word base form as unicode or None

Example:

```
>>> dic.id_base('123643@gen12')
u'zamek'
```

4.6 `word_base` method

`PyDicManager.`**`word_base`**(*word*)

Returns unique word base forms for a given word

Parameters **word** (*unicode*) – a word form

Returns unique list of forms as unicode

Example:

```
>>> dic.word_base(u'zamkowi')
[u'zamek', u'zamkowy']
```

Warning: Querying is case-insensitive.

Warning: As you can see there can be more than one inflectional vector that matches a given word. Therefore this function always return list of lists.

Note: Elements on that list are unique.

CHAPTER 5

PyDicId identifiers

PyDicId allows to precisely identify a single dictionary lexem. Identifier is build from two parts: * sequential number (id property), * dictionary name (dict property).

Eg.: “123@sjp”, “1004@slang-dict”, etc.

PyDicId can be created in two simple ways, by constructing PyDic object from str or unicode:

```
>>> from pydic import PyDicId
>>> PyDicId("123@sjp")
```

or by explicitly defining id and dict properties:

```
>>> PyDicId(123, "sjp")
```

PyDicId are fully interoperable with corresponding strings, so there is a great variety what you can do with that:

```
>>> word = PyDicId('123@sjp')
>>> word == '123@sjp' and word == u'123@sjp'
True
>>> word.id
123
>>> word.dict
u'sjp'
```

In fact whenever PyDicId is required you can also populate string id and it will be automatically converted to corresponding PyDicId.

Having access to PyDicId properties (rather then bare string id) is very convenient for processing multidictionary cases:

```
>>> words = [PyDicId(u'123@sjp'), PyDicId(u'124@sjp'), PyDicId(u'123@slang')]
>>> filter(lambda w: w.dict == 'sjp', words)
[PyDicId(u'123@sjp'), PyDicId(u'124@sjp')]
```

Note: Please note that PyDicId are very loosely coupled with dictionaries itself. For example you can easily create some random identifier that does not belong to any dictionary.

CHAPTER 6

Stemmer

Stemming is the process for reducing inflected (or sometimes derived) words to their stem, base or root form—generally a written word form.

For now stemming module of PyDic allow to index base words from a single dictionary and make an inflection of any word list, building in this way a new dictionaries from new, possibly unknown words.

Warning: Stemmer is currently under heavy development.

6.1 Syntax

```
pydic_stemmer.py --help
usage: pydic_stemmer.py [-h] [-d DELIMITER] -f DICTIONARY_FILE [-t OUTPUT]
                        [-b] [-v]
                        [FILE]

Makes inflection of a flat text file with words.

positional arguments:
  FILE                  filename to process

optional arguments:
  -h, --help            show this help message and exit
  -d DELIMITER, --delimiter DELIMITER
  -f DICTIONARY_FILE, --dictionary-file DICTIONARY_FILE
                        path to file with text dictionary
  -t OUTPUT, --output OUTPUT
                        output file name
  -b, --base-forms      only base forms
  -v, --verbose         debug verbose mode
```

6.2 Input data format

A list of base forms of unknown words, eg.

```
$ cat new.txt
supermegapojazd
oktokrążownik
```

6.3 Inflecting words

```
$ pydic_stemmer.py -f sjp.pydic new.txt
supermegapojazd,supermegapojazdach,supermegapojazdami,supermegapojazdem,supermegapojazdom,supermegapo
oktokrążownik,oktokrążownika,oktokrążownikach,oktokrążownikami,oktokrążowniki,oktokrążownikiem,oktok
```

CHAPTER 7

Indices and tables

- *genindex*
- *modindex*
- *search*