bandicoot Documentation

Release 0.1

Yves-Alexandre de Montjoye, Florent Robic, Luc Rocher

August 14, 2014

CONTENTS

bandicoot is a python toolbox to extract meaningful features from metadata. It has been created by Yves-Alexandre de Montjoye, Luc Rocher, Florent Robic, and Alex Pentland at the MIT Media Lab.

Features computed by bandicoot have already been used to release data as part of Orange D4D Challenge, to predict personality, and for customer segmentation

If you use bandicoot for your research please cite it:

Note: de Montjoye, Y.-A.*, Quoidbach J.*, Robic F.*, Pentland A., Predicting people personality using novel mobile phone-based metrics. International Conference on Social Computing, Behavioral-Cultural Modeling, & Prediction, Washington, USA (2013).

CHAPTER

ONE

QUICK START

1.1 Installation

You can install bandicoot by running one command from a terminal:

```
python setup.py install
```

If you don't have write permission to the global site-packages directory, you can install bandicoot only for the current user, using:

```
python setup.py install --user
```

1.2 Loading data

bandicoot takes two files per user as standard input: the records files and the attribute file.

1.2.1 Records and places files

The records file contains the call, text and mobility records of an individual.

interaction	direction	correspondent_id	datetime	call_duration	place_id
call	in	8f8ad28de134	2012-05-20 20:30:37	137	13084
call	out	fe01d67aeccd	2012-05-20 20:31:42	542	13084
text	in	c8f538f1ccb2	2012-05-20 21:10:31		13087

records.csv:

```
interaction,direction,correspondent_id,datetime,call_duration,antenna_id
call,in,8f8ad28de134,2012-05-20 20:30:37,137,13084
call,out,fe01d67aeccd,2012-05-20 20:31:42,542,13084
text,in,c8f538f1ccb2,2012-05-20 21:10:31,,13087
```

while the places file contains the latitude and longitude coordinates of the places

place_id	latitude	longitude
13084	42.360888	-71.0877297
13087	42.367709	-71.107692

places.csv:

```
place_id,latitude,longitude
13084,42.360888,-71.0877297
13087,42.367709,-71.107692
```

They can be loaded as csv, with the following headers:

```
>>> B = bc.read_csv('records.csv', places_path='places.csv')
```

1.2.2 Attribute file

The attribute file contains information about the individual that can be used for example to compute the ego-network assortativity or clustering coefficient. Any attribute can be loaded and values can be string, int, or float. bandicoot predefines a few keys such as individual_id, gender, or subscriber.

key	value
individual_id	7atr8f53fg41
gender	male
is_subscriber	True
age	42

It can be loaded as a csv, with the following header

```
key,value
individual_id,7atr8f53fg41
gender,male
is_subscriber,True
age,42
```

Attributes is optional and can be loaded at the same time as the records using `read_csv`.

```
>>> B = bc.read_csv('records.csv', places_path='places.csv', attributes_path='attributes.csv')
```

1.3 Computing features

By default, bandicoot computes the features on a weekly basis and returns the mean and standard error on the mean (sem) in a nested dictionary. If needed, the dictionary can be flatten using `bc.helper.flatten`.

For example, say the individual mean text response rate is 0.5 for the first week, 0.32 for the second week, and 0.52 for the third week. Calling `bc.behavior.response_rate_text` will return the mean and sem over all the available weeks:

```
>>> bc.behavior.response_rate_text(B)
{'text': {'mean': 0.446, 'sem': 0.063}}
```

1.3.1 Interaction type: call, text, or call and text

Some features can be computed on both *call*, *text*, or *callandtext*. This can be specified with the argument `interactions`

```
>>> bc.diversity.number_of_contacts(B, interactions=['call','text'])
{'call': {'mean': {'mean': 15.2, 'sem': 0.32}},
 'text': {'mean': {'mean': 7, 'sem': 0.23}}}
```

By default, features are–when possible–computed on *callandtext*. Some features can also be computed on *mobility*.

1.3.2 Summary statistics

Some features such as `interevent_time` return a list per week. For example, *[1612, 1229, 451, 122, 897, 2217, 3479, 3038, 3488, 458]* for the first week and *[488, 3484, 945, 3026, 3469, 1209, 1879, 2703, 3151, 1157]* for the second week.

By default, bandicoot will compute the mean value and return it:

```
>>> bc.diversity.interevent_time(B, interactions=['call'])
{'call': {'mean': {'mean': 1925.099, 'sem': 226.0}}}
```

Other summary statistics can, however, be returned. If passed `summary='extended'`, the median and the standard deviation will also be returned:

```
>>> bc.diversity.interevent_time(B, interactions=['call'], summary='extended')
{'call': {'mean': {'mean': 1925.099, 'sem': 226.0},
  'median': {'mean': 1855.75, 'sem': 435.25},
  'std': {'mean': 1152.121, 'sem': 67.975}}}
```

`summary='all'` will also return the fourth standardized moment of the distribution, the kurtosis (a measure of peakness), and the skewness of the distribution.

Finally, if passed `summary=None` the full distributions will be returned in a list of list:

```
>>> bc.diversity.interevent_time(B, interactions=['call'], summary=None)
{'call': [[1612, 1229, 451, 122, 897, 2217, 3479, 3038, 3488, 458],
  [488, 3484, 945, 3026, 3469, 1209, 1879, 2703, 3151, 1157]]}
```

summary	returns
mean (default)	mean
extended	mean, median, std
all	mean, median, std, kurtosis, skewness
None	the full distribution

1.4 Helper functions

bandicoot provides a set of helper functions under `bc.helper`.

1.4.1 Flattening dictionaries

`bc.helper.flatten` flattens bandicoot dictionaries as follow:

```
>>> contacts = bc.diversity.interevent_time(B, interactions=['call'], summary='extended')
>>> contacts
{'call': {'mean': {'mean': 1925.099, 'sem': 226.0},
  'median': {'mean': 1855.75, 'sem': 435.25},
  'std': {'mean': 1152.121, 'sem': 67.975}}}
>>> bc.helper.flatten(contacts)
{'call_mean_mean': 1925.099,
 'call_mean_sem': 226.0,
 'call_median_mean': 1855.75,
 'call_median_sem': 435.25,
 'call_std_mean': 1152.121,
 'call_std_sem': 67.975}
```

1.4.2 Bulk

The bulk function computes the default bandicoot features on `call`, `text` and `callandtext` returning the weekly mean and sem:

```
>>> bc.helper.all(B)
{'number_of_contacts': {'call': {'mean': {'mean': 15.2, 'sem': 0.32}},
  'callandtext': {'mean': 21.2, 'sem': 1.56},
  'text': {'mean': {'mean': 7, 'sem': 0.23}}},
 'response_rate_text': {'text': {'mean': 0.446, 'sem': 0.063}}}
```

CHAPTER

TWO

REFERENCE

2.1 core

2.1.1 User class

class bandicoot.core.**User**

Data structure storing all the call, text or mobility records of the user.

compute_home()

Return the place where the user spends most of his time at night.

describe()

Generate a short description of the object.

Notes

The summary is directly sent to the standard output.

Examples

```
>>> import bandicoot as bc
>>> user = bc.User()
>>> user.records = bc.tests.generate_user.random_burst(5)
>>> user.describe()
[x] 5 records from 2014-01-01 10:41:00 to 2014-01-01 11:21:00
    5 contacts
[x] 1 attribute
```

records

Can be used to get or set the list of user's records.

If the records are modified, the start and end time as well as the infered house properties of the object User are recomputed.

2.1.2 Records

class bandicoot.core.**Record**

call_duration

Alias for field number 4

correspondent_id
Alias for field number 2

datetime
Alias for field number 3

direction
Alias for field number 1

interaction
Alias for field number 0

position
Alias for field number 5

2.2 IO

2.2.1 Submodules

bandicoot.io.parsers module

read_csv(records_path[, attributes_path])	Load CSV files of records and attributes into a new user.
read_orange([record_path])	Load data from a CSV file with *orange* format:
read_telenor(incoming_cdr, outgoing_cdr, ...)	Load data from CSV files using the *Telenor* format.

read_csv

bandicoot.io.parsers.**read_csv**(*records_path*, *attributes_path=None*)
Load CSV files of records and attributes into a new user.

Parameters **records_path** : str

Path of the CSV file containing all the records of a user. Records are stored in the `user` property `records()`.

attributes_path : str, optional

Path of the CSV file containing (key, value) attributes like age or gender. Attributes can be helpful to compute specific metrics.

Examples

```
>>> user = bandicoot.read_csv('sample_records.csv')
>>> print(len(user.records))
10

>>> user = bandicoot.read_csv('sample_records.csv', 'sample_attributes.csv')
>>> print user.attributes['age']
25
```

read_orange

bandicoot.io.parsers.**read_orange**(*record_path=None*)

Load data from a CSV file with *orange* format:

```
call_record_type;basic_service;user_msisdn;call_partner_identity;datetime;call_duratio
```

`basic_service` takes one of the following values:

- 11: telephony;
- 12: emergency calls;
- 21: short message (in)
- 22: short message (out)

Parameters **record_path** : str or iterator, optional

If `record_path` is a string, the function will load the CSV file at the path `record_path`. If the parameter is an iterator, records will be loaded directly, as an ordered list in the *orange* format. If no parameter is included, `read_orange` will load records from the standard input `sys.stdin`

read_telenor

bandicoot.io.parsers.**read_telenor**(*incoming_cdr*, *outgoing_cdr*, *cell_towers*)

Load data from CSV files using the *Telenor* format.

2.3 behavior

active_days(_user[, method, interaction])	Returns the number of active days, when the user makes at least one a
duration_of_calls(_user[, method, interaction])	Return the average, median and standard deviation of the duration of c
number_of_interactions(_user[, method, ...])	Returns the total number of interactions the user had.
percent_initiated(_user[, method, interaction])	Return the percentage of initiated calls.
percent_nocturnal(_user[, method, interaction])	Return the percentage of interactions at night.
percentage_initiated_conversation(_user[, ...])	Compute the percentage of initiated conversations (a suite of interacti
response_delay_text(_user[, method, interaction])	Compute the mean, standard deviation and median of the distribution
response_rate_text(_user[, method, interaction])	Compute the average response rate of the user across all its contacts.

2.3.1 active_days

bandicoot.behavior.**active_days**(*_user*, *method='weekly'*, *interaction='all'*)

Returns the number of active days, when the user makes at least one action: send or received a text, initiated or received a call, has a mobility point.

2.3.2 duration_of_calls

bandicoot.behavior.**duration_of_calls**(*_user*, *method='weekly'*, *interaction='call'*)

Return the average, median and standard deviation of the duration of calls.

2.3.3 number_of_interactions

bandicoot.behavior.**number_of_interactions**(*_user*, *method='weekly'*, *interaction=None*)
Returns the total number of interactions the user had.

2.3.4 percent_initiated

bandicoot.behavior.**percent_initiated**(*_user*, *method='weekly'*, *interaction='call'*)
Return the percentage of initiated calls.

2.3.5 percent_nocturnal

bandicoot.behavior.**percent_nocturnal**(*_user*, *method='weekly'*, *interaction=None*)
Return the percentage of interactions at night.

By default, night is defined as 7pm-7am but can be changed in `user.night_start` and `user_night_end`.

2.3.6 percentage_initiated_conversation

bandicoot.behavior.**percentage_initiated_conversation**(*_user*, *method='weekly'*, *interaction='all'*)
Compute the percentage of initiated conversations (a suite of interactions spaced of others by more than one hour).

2.3.7 response_delay_text

bandicoot.behavior.**response_delay_text**(*_user*, *method='weekly'*, *interaction='all'*)
Compute the mean, standard deviation and median of the distribution of response delays inside conversations (for all responses of the user to an incoming text).

The following suite of messages defines four conversations (`I` for an incoming text, `O` for an outgoing text, `-` for a five minutes delay):

```
I-O--I----O I---I--O--I I--I-I---I-I O--O----I-O
```

The response delays are :

- 5 minutes, 20 minutes for the first conversation;
- 10 minutes for the second;
- none for the third;
- 5 minutes for the last.

The function will return the mean, standard deviation and median of the distribution `[5, 5, 10, 20]`, winsorized at 99%, as an `MSM` object:

```
MSM(mean=10.0, std=7.0710678118654755, median=7.5)
```

Notes

Because conversations are grouped, with a time limit of one hour, the response delay cannot be superior to one hour.

The distribution is winsorized at 99% to remove any effect of outliers.

2.3.8 response_rate_text

bandicoot.behavior.**response_rate_text**(*_user*, *method='weekly'*, *interaction='all'*)

Compute the average response rate of the user across all its contacts.

The response rate is the ratio of conversations which started with at least one response from the user (among all conversations starting with a received text).

The following suite of messages defines four conversations (`I` for an incoming text, `O` for an outgoing text):

```
I-O-I-O
I-I-O-I
I-I-I-I-I
O-O-I-O
```

Only the first three conversations start with an incoming text. The first and second ones have at least one response. The response ratio will be 2/3.

Notes

A user can respond to a conversation with a call.

2.4 diversity

interevents_time(_user[, method, interaction])	Computes the interevent time between all the records of the user and retu
number_of_contacts(_user[, method, interaction])	Returns the number of contacts the user interacted with via calls of texts.
entropy_of_contacts(_user[, method, interaction])	Returns the entropy of a user's contacts.
interaction_per_contact(_user[, method, ...])	Computes the number of interactions a user has with its contacts.

2.4.1 interevents_time

bandicoot.diversity.**interevents_time**(*_user*, *method='weekly'*, *interaction=None*)

Computes the interevent time between all the records of the user and returns its mean, standard deviation, and median.

2.4.2 number_of_contacts

bandicoot.diversity.**number_of_contacts**(*_user*, *method='weekly'*, *interaction=None*)

Returns the number of contacts the user interacted with via calls of texts.

2.4.3 entropy_of_contacts

bandicoot.diversity.**entropy_of_contacts**(*_user*, *method='weekly'*, *interaction=None*)
Returns the entropy of a user's contacts.

By default it returns the entropy of a user call contacts and text contacts.

2.4.4 interaction_per_contact

bandicoot.diversity.**interaction_per_contact**(*_user*, *method='weekly'*, *interaction=None*)
Computes the number of interactions a user has with its contacts.

By default it returns the mean, median, and standard deviation of a user's number of interaction for call and text contacts.

2.5 spatial

percent_at_home(_user[, method, interaction])	Returns the percentage of interactions the user had while at home.
radius_of_gyration(_user[, method, interaction])	Returns the radius of gyration [GON2008]
entropy_places(_user[, method, interaction])	Returns the entropy of the distribution of visited places.
number_of_places(_user[, method, interaction])	Returns the number of unique places visited.

2.5.1 percent_at_home

bandicoot.spatial.**percent_at_home**(*_user*, *method='weekly'*, *interaction='all'*)
Returns the percentage of interactions the user had while at home.

Notes

The position of the home is computed by `User.compute_home()`. If no home can be found, the percentage at home will be None.

2.5.2 radius_of_gyration

bandicoot.spatial.**radius_of_gyration**(*_user*, *method='weekly'*, *interaction='all'*)
Returns the radius of gyration [GON2008]

The radius of gyration is the *equivalent distance* of the mass from the center of gravity, for all visited places.

2.5.3 entropy_places

bandicoot.spatial.**entropy_places**(*_user*, *method='weekly'*, *interaction='all'*)
Returns the entropy of the distribution of visited places.

2.5.4 number_of_places

bandicoot.spatial.**number_of_places**(*_user*, *method='weekly'*, *interaction='all'*)
Returns the number of unique places visited.

2.6 helper

2.6.1 group submodule

group_records(records[, interactions, method])	Group records by year and week number.
grouping([f, user, interaction])	
statistics(aggregated[, _fun_call])	Return statistics (mean, standard error, standard error of the mean and median)

group_records

bandicoot.helper.group.**group_records**(*records*, *interactions=None*, *method='weekly'*)

Group records by year and week number.

Parameters **records** : iterator

An iterator over records

method : {'weekly', 'biweekly', 'monthly'}, default 'weekly'

- weekly: group records by year and week
- biweekly: group records by chunks of two weeks, starting at the first week
- monthly: group records by year and month

grouping

bandicoot.helper.group.**grouping**(*f=None*, *user=False*, *interaction=None*)

statistics

bandicoot.helper.group.**statistics**(*aggregated*, *_fun_call=None*)

Return statistics (mean, standard error, standard error of the mean and median) on aggregated metrics.

Examples

Given a list of integers or floating point numbers, `statistics` computes the mean and standard error of the mean.

```
>>> statistics([0, 1, 2, 3])
{'mean': 1.0, 'sem': 0.5}
```

Given a list of `MSM` tuples `(mean, std, median)`, the function will returns the mean and standard error of the mean for each attributes of the tuples.

```
>>> msm_1 = MSM(10, 2, 10)
>>> msm_2 = MSM(12, 2, 12)
>>> statistics([msm_1, msm_2])
{'mean': {'mean': 11.0, 'sem': 1.0},
 'median': {'mean': 11.0, 'sem': 1.0},
 'std': {'mean': 2.0, 'sem': 0.0}}
```

2.6.2 utils submodule

MSM	
flatten(d[, parent_key, separator])	Flatten a nested dictionary.
pairwise(iterable)	s -> (s0,s1), (s1,s2), (s2, s3), ...
mean(data[, limit, winsorize])	Return a winsorized arithmetic mean of data.
median(data)	Return the median of numeric data, unsing the "mean of middle two" method.
std(data[, winsorize, limit])	
sem(data)	Standard error of the mean.
msm(data, winsorize[, limit])	Returns a tuple containing the winsorized mean, standard deviation and median of the first
entropy(data)	Compute the Shannon entropy, a measure of uncertainty.
great_circle_distance(pt1, pt2)	
all(user[, method])	Return a dictionary containing all bandicoot metrics for a given user.

bandicoot.helper.utils.MSM

class bandicoot.helper.utils.**MSM**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

count(...)	
index((value, [start, ...)	Raises ValueError if the value is not present.

Attributes

mean	Alias for field number 0
median	Alias for field number 2
std	Alias for field number 1

flatten

bandicoot.helper.utils.**flatten**(*d*, *parent_key=''*, *separator='_'*)
Flatten a nested dictionary.

Parameters d: dict_like :

Dictionnary to flatten.

parent_key: string, optional :

Concatenated names of the parent keys.

separator: string, optional :

Separator between the names of the each key. The default separator is '_'.

Examples

```
>>> d = {'alpha': 1, 'beta': {'a': 10, 'b': 42}}
>>> flatten(d) == {'alpha': 1, 'beta_a': 10, 'beta_b': 42}
True
>>> flatten(d, separator='.') == {'alpha': 1, 'beta.a': 10, 'beta.b': 42}
True
```

pairwise

bandicoot.helper.utils.**pairwise**(*iterable*)

s -> (s0,s1), (s1,s2), (s2, s3), ...

mean

bandicoot.helper.utils.**mean**(*data*, *limit=0.99*, *winsorize=True*)

Return a winsorized arithmetic mean of `data`. If `limit=1`, it returns the classical mean.

Examples

```
>>> mean([1, 2, 3, 4, 4], limit=1)
2.8
```

In the following example, the first and last value of the list are replaced by their next value.

```
>>> mean([1, 2, 3, 4, 5, 6, 7, 8, 9, 100], limit=0.8)
5.5
>>> mean([1, 2, 3, 4, 5, 6, 7, 8, 9, 100], limit=1.)
14.5
```

median

bandicoot.helper.utils.**median**(*data*)

Return the median of numeric data, unsing the "mean of middle two" method. If `data` is empty, `0` is returned.

Examples

```
>>> median([1, 3, 5])
3.0
```

When the number of data points is even, the median is interpolated: >>> median([1, 3, 5, 7]) 4.0

std

bandicoot.helper.utils.**std**(*data*, *winsorize=True*, *limit=0.99*)

sem

bandicoot.helper.utils.**sem**(*data*)

Standard error of the mean.

bandicoot.helper.utils.MSM

class bandicoot.helper.utils.**MSM**

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

count(...)	
index((value, [start, ...)	Raises ValueError if the value is not present.

Attributes

mean	Alias for field number 0
median	Alias for field number 2
std	Alias for field number 1

entropy

bandicoot.helper.utils.**entropy**(*data*)
Compute the Shannon entropy, a measure of uncertainty.

great_circle_distance

bandicoot.helper.utils.**great_circle_distance**(*pt1*, *pt2*)

all

bandicoot.helper.utils.**all**(*user*, *method='weekly'*)
Return a dictionary containing all bandicoot metrics for a given user.

2.7 tests

2.7.1 Submodules

bandicoot.tests.generate_user module

bandicoot.tests.generate_user.**random_burst**(*count*, *delta=datetime.timedelta(0, 600)*, ***kwargs*)

bandicoot.tests.generate_user.**random_record**(***kwargs*)

bandicoot.tests.generate_user.**random_records**(*n*, *antennas*, *number_of_users=150*, *ingoing=0.7*, *percent_text=0.3*, *rate=0.0001*)

bandicoot.tests.test module

bandicoot.tests.test_sequences module

class bandicoot.tests.test_sequences.**InterEventsTests**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

test_empty_interevents()

test_sequence_interevents()

CHAPTER

THREE

EXTENDING BANDICOOT

3.1 The user object

The user object is composed of a list of records, a dictionary of attributes, and object's attributes.

3.1.1 List of records

A record is stored as a named tuple by the class `Record`:

name	type	description
interaction	string [required]	`'call'`, `'text'`, or `'mobility'`
direction	string	whether the user was called (`'in'`) or was the one calling (`'out'`)
correspondent_id	string	identifier of the correspondent
datetime	datetime	timestamp of the record
call_duration	interaction	duration of the call in seconds
position	list	a list with `'is_place'` (True or False) and then a either a latlon tuple or the identifier of a place as string. For example, [False, (42.360888,-71.0877297)] or [True, '13084']

A user's records are stored as a list, and can be accessed or modified with the property `User.records`.

3.1.2 Dictionary of user attributes

User attributes can be loaded at the same time as his records. Attributes are stored in a dictionary that can be access by `User.attributes`:

```
>>> user.attributes['age'] = 42
>>> user.attributes['likes_trains'] = True
```

bandicoot has _reserved_ names for a few attributes:

keys	type	description
individual_id	string	the user identifier, this is required for networked users
gender	string	can be `male` or `female`
age	int	age of the user

3.1.3 Object attributes

Object attributes are created by bandicoot when the user is loaded using `read_csv`.

keys	type	description
has_call	bool	whether call records have been loaded
has_text	bool	whether text records have been loaded
has_mobility_places	bool	whether places records have been loaded
has_mobility_gps	bool	whether gps records have been loaded
starttime	date-time	time of the first record
endtime	date-time	time of the last record
places_dict	dict	dictionary of places with place_id as keys and latlon tuples
home_places	dict	dictionary of places for home location of the user and their weights {'13084': 5, '13087': 1}
home_gps	tuple	a latlon tuple for home location of the user

3.2 Conversations

A conversation is a suite of text messages between two users, spaced by less than one hour. Bandicoot computes several metrics on conversations : `response_rate_text()`, `percentage_initiated_conversation()`...

A call always closes an open conversation.

Index

- *modindex*

[GON2008] Gonzalez, M. C., Hidalgo, C. A., & Barabasi, A. L. (2008). Understanding individual human mobility patterns. Nature, 453(7196), 779-782.

b

b

Symbols

A

B

C

D

E

F

G

I

M

N

P

R

S

T

U