CLK hash Documentation

Release 0.9.0

N1 Analytics

Apr 23, 2018

Contents:

CHAPTER 1

API Documentation

1.1 Bloom filter

Generate a Bloom filter

clkhash.bloomfilter.**calculate_bloom_filters**(*dataset*, *schema*, *keys*, *xor_folds=0*)

Parameters

- **dataset** – A list of indexable records.
- **schema** – An iterable of identifier types.
- **keys** – A tuple of two lists of secret keys used in the HMAC.
- **xor_folds** – number of XOR folds to perform

Returns List of bloom filters as 3-tuples, each containing bloom filter (bitarray), record first element - usually index, bitcount (int)

clkhash.bloomfilter.**crypto_bloom_filter**(*record*, *tokenizers*, *keys1*, *keys2*, *xor_folds=0*, *l=1024*, *k=30*)

Makes a Bloom filter from a record with given tokenizers and lists of keys.

Using the method from http://www.record-linkage.de/-download=wp-grlc-2011-02.pdf

Parameters

- **record** – plaintext record tuple. E.g. (index, name, dob, gender)
- **tokenizers** – A list of IdentifierType tokenizers (one for each record element)
- **keys1** – list of keys for first hash function as list of bytes
- **keys2** – list of keys for second hash function as list of bytes
- **xor_folds** – number of XOR folds to perform
- **l** – length of the Bloom filter in number of bits
- **k** – number of hash functions to use per element

Returns 3-tuple: - bloom filter for record as a bitarray - first element of record (usually an index) - number of bits set in the bloomfilter

clkhash.bloomfilter.**double_hash_encode_ngrams**(*ngrams*, *key_sha1*, *key_md5*, *k*, *l*)
computes the double hash encoding of the provided ngrams with the given keys.

Using the method from http://www.record-linkage.de/-download=wp-grlc-2011-02.pdf

Parameters

- **ngrams** – list of n-grams to be encoded
- **key_sha1** – hmac secret keys for sha1 as bytes
- **key_md5** – hmac secret keys for md5 as bytes
- **k** – number of hash functions to use per element of the ngrams
- **l** – length of the output bitarray

Returns bitarray of length l with the bits set which correspond to the encoding of the ngrams

clkhash.bloomfilter.**fold_xor**(*bloomfilter*, *folds*)
Performs XOR folding on a Bloom filter.

If the length of the original Bloom filter is n and we perform r folds, then the length of the resulting filter is n / 2 ** r.

Parameters

- **bloomfilter** – Bloom filter to fold
- **folds** – number of folds

Returns folded bloom filter

clkhash.bloomfilter.**serialize_bitarray**(*ba*)
Serialize a bitarray (bloomfilter)

clkhash.bloomfilter.**stream_bloom_filters**(*dataset*, *schema_types*, *keys*, *xor_folds=0*)
Yield bloom filters

Parameters

- **dataset** – An iterable of indexable records.
- **schema_types** – An iterable of identifier type names.
- **keys** – A tuple of two lists of secret keys used in the HMAC.
- **xor_folds** – number of XOR folds to perform

Returns Yields bloom filters as 3-tuples

1.2 CLK

Generate CLK from CSV file

clkhash.clk.**chunks**(*l*, *n*)
Yield successive n-sized chunks from l.

clkhash.clk.**generate_clk_from_csv**(*input*, *keys*, *schema_types*, *no_header=False*, *progress_bar=True*, *xor_folds=0*)

clkhash.clk.**generate_clks**(*pii_data*, *schema_types*, *key_lists*, *xor_folds*, *callback=None*)

clkhash.clk.**hash_and_serialize_chunk**(*chunk_pii_data*, *schema_types*, *keys*, *xor_folds*)
Generate Bloom filters (ie hash) from chunks of PII then serialize the generated Bloom filters.

Parameters

- **chunk_pii_data** – An iterable of indexable records.
- **schema_types** – An iterable of identifier type names.
- **keys** – A tuple of two lists of secret keys used in the HMAC.
- **xor_folds** – Number of XOR folds to perform. Each fold halves the hash length.

Returns A list of serialized Bloom filters

1.3 identifier types

Convert PII to tokens

clkhash.identifier_types.**identifier_type_from_description**(*schema_object*)
Convert a dictionary describing a feature into an IdentifierType

Parameters **schema_object** –

Returns An IdentifierType

1.4 IdentifierType

class clkhash.identifier_types.**IdentifierType**(*unigram=False*, *weight=1*, ***kwargs*)
Bases: object

Base class used for all identifier types.

Required to provide a mapping of schema to hash type uni-gram or bi-gram.

__call__(*entry*)
Call self as a function.

__init__(*unigram=False*, *weight=1*, ***kwargs*)

Parameters

- **unigram** (*bool*) – Use uni-gram instead of using bi-grams
- **weight** (*float*) – adjusts the "importance" of this identifier in the Bloom filter. Can be set to zero to skip
- **kwargs** – Extra keyword arguments passed to the tokenizer

---

Note: For each n-gram of an identifier, we compute k different indices in the Bloom filter which will be set to true. There is a global $k_{default}$ value, and the k value for each identifier is computed as

$$k = weight * k_{default},$$

rounded to the nearest integer.

Reasons why you might want to set weights:

- Long identifiers like street name will produce a lot more n-grams than small identifiers like zip code. Thus street name will flip more bits in the Bloom filter and will have a bigger influence in the overall matching score.
- The matching might produce better results if identifiers that are stable and / or have low error rates are given higher prominence in the Bloom filter.

__weakref__
list of weak references to the object (if defined)

1.5 key derivation

class clkhash.key_derivation.**HKDFconfig**(*master_secret*, *salt=None*, *info=None*, *hash_algo='SHA256'*)

Bases: object

static check_is_bytes(*value*)

static check_is_bytes_or_none(*value*)

supported_hash_algos = ('SHA256', 'SHA512')

clkhash.key_derivation.**generate_key_lists**(*master_secrets*, *num_identifier*, *key_size=64*, *salt=None*, *info=None*, *kdf='HKDF'*)

Generates a derived key for each identifier for each master secret using a key derivation function (KDF).

The only supported key derivation function for now is 'HKDF'.

The previous key usage can be reproduced by setting kdf to 'legacy'. This is highly discouraged, as this strategy will map the same n-grams in different identifier to the same bits in the Bloom filter and thus does not lead to good results.

Parameters

- **master_secrets** – a list of master secrets (either as bytes or strings)
- **num_identifier** – the number of identifiers
- **key_size** – the size of the derived keys
- **salt** – salt for the KDF as bytes
- **info** – optional context and application specific information as bytes
- **kdf** – the key derivation function algorithm to use

Returns The derived keys. First dimension is the same as master_secrets, second dimension is of size num_identifier. A key is represented as bytes.

clkhash.key_derivation.**hkdf**(*hkdf_config*, *num_keys*, *key_size=64*)

Executes the HKDF key derivation function as described in rfc5869 to derive *num_keys* keys of size *key_size* from the master_secret.

Parameters

- **hkdf_config** – an HKDFconfig object containing the configuration for the HKDF.
- **num_keys** – the number of keys the kdf should produce
- **key_size** – the size of the produced keys

Returns Derived keys

1.6 random names

Module to produce a dataset of names, genders and dates of birth and manipulate that list

Currently very simple and not realistic. Additional functions for manipulating the list of names - producing reordered and subset lists with a specific overlap

ClassList class - generate a list of length n of [id, name, dob, gender] lists

TODO: Get age distribution right by using a mortality table TODO: Get first name distributions right by using distributions TODO: Generate realistic errors TODO: Add RESTfull api to generate reasonable name data as requested

class clkhash.randomnames.**NameList**(*n*)

Bases: object

List of randomly generated names

generate_random_person(*n*)

Generator that yields details on a person with plausible name, sex and age.

Yields Generated data for one person tuple - (id: int, name: str('First Last'), birthdate: str('DD/MM/YYYY'), sex: str('M' | 'F'))

generate_subsets(*sz*, *overlap=0.8*)

Generate a pair of subsets of the name list with a specified overlap

Parameters

- **sz** – length of subsets to generate
- **overlap** – fraction of the subsets that should have the same names in them

Returns 2-tuple of lists of subsets

load_names()

This function loads a name database into globals firstNames and lastNames

initial version uses data files from http://www.quietaffiliate.com/free-first-name-and-last-name-databases-csv-and-sql/

schema = [{'identifier': 'INDEX'}, {'identifier': 'NAME freetext'}, {'identifier':

schema_types

clkhash.randomnames.**load_csv_data**(*resource_name*)

Loads a specified data file as csv and returns the first column as a Python list

clkhash.randomnames.**random_date**(*start*, *end*)

This function will return a random datetime between two datetime objects.

Parameters

- **start** – datetime of start
- **end** – datetime of end

Returns random datetime between start and end

clkhash.randomnames.**save_csv**(*data*, *schema*, *file*)

Output generated data as csv with header.

Parameters

- **data** – An iterable of tuples containing raw data.

- **schema** – Iterable of schema definition dicts
- **file** – A writeable stream in which to write the csv

1.7 schema

clkhash.schema.**get_schema_types**(*schema*)

clkhash.schema.**load_schema**(*schema_file*)

1.8 tokenizer

Functions to tokenize words (PII)

clkhash.tokenizer.**bigramlist**(*word*, *toremove=None*)

Make bigrams from word with pre- and ap-pended spaces

s -> [‘ ‘ + s0, s0 + s1, s1 + s2, .. sN + ‘ ‘]

Parameters

- **word** – string to make bigrams from
- **toremove** – List of strings to remove before construction

Returns list of bigrams as strings

clkhash.tokenizer.**positional_unigrams**(*instr*)

Make positional unigrams from a word.

E.g. 1987 -> [“1 1”, “2 9”, “3 8”, “4 7”]

Parameters **instr** – input string

Returns list of strings with unigrams

clkhash.tokenizer.**unigramlist**(*instr*, *toremove=None*, *positional=False*)

Make 1-grams (unigrams) from a word, possibly excluding particular substrings

Parameters

- **instr** – input string
- **toremove** – Iterable of strings to remove

Returns list of strings with unigrams

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

Symbols

B

C

D

F

G

H

I

L

N

P

R

S

U