EHRcorral Documentation

Release 0.0.3

Nikhil Haas

December 18, 2015

Contents

EHRCorral creates a master patient index (MPI) by matching and linking electronic medical records.

EHR noun (electronic healtchare record) 1. a health record in digital format: *the patient's medical diagnosis was entered into the EHR.*

corral |kral| verb (corrals, corralling, corralled) [with obj.] 1. gather together and confine (a group of people or things): *the organizers were corralling the crowd into marching formation.*

CHAPTER 1

EHRcorral

EHRCorral matches, links, and de-duplicates electronic medical records for the purpose of creating a master patient index (MPI).

- Free software: ISC license
- Documentation: https://ehrcorral.readthedocs.org.

1.1 Quick Start

CHAPTER 2

Overview

Significant effort has been put into developing record-linkage algorithms using deterministic, probabilistic, or machine learning methods, or a combination of approaches [1] [2] [3]. EHRcorral takes a probabilistic approach, wherein certain fields are weighted based on their match-level, which is determined using numerical or lexical analysis in the context of two records or the entire set of records. A composite probability of two records matching is calculated and if the probability is above a threshold value the records are linked. Several pre-processing steps are often taken to reduce the computational requirements and attempt to increase the sensitivity and specificity of the algorithm.

2.1 Precedents

Purely deterministic models, which attempt to find identical values in certain fields, are unideal for many healthcare data sets [4]. Keying errors, misspellings, and transpositions of first name and last name are all too common in EHRs [5] [6], and some institutions are only able to record minimal identifying information about patients, such as is often the case with transient, homeless, and under-served populations. This makes it difficult to identify a field or fields which can reliably be matched exactly across records.

Machine learning algorithms, such as neural networks, can be used for matching, with pros and cons compared to other approaches [7]. However, while machine learning is becoming more common in many fields as computational units become cheaper, most of these algorithms require some method of training in order to identify a "pattern" and develop a specific algorithm to be applied on future records for linkage. This training might entail feeding in a large data set where record links have already been identified, or training the algorithm as it is developed.

A probabilistic method can run immediately on a data set without training data and identify record linkages with surprising sensitivity and specificity when the right settings are used. The OX-LINK system, which was developed to match 58 million healthcare records spanning from the 1960s to the '90s, achieved a false positive rate of 0.20%—0.30% and a false negative rate from 1.0%—3.0% on several hundred thousand records [8]. This system uses a combination of probabilistic, weighted matching, lexical analysis, phonemic blocking, and manual review. Recent publications also suggest that high sensitivity can be achieved with probabilistic methods, even in the context of error-prone data.

[1] Gu, Lifang, et al. "Record linkage: Current practice and future directions." CSIRO Mathematical and Information Sciences Technical Report 3 (2003): 83.

[2] Winkler, William E. "The state of record linkage and current research problems." Statistical Research Division, US Census Bureau. 1999.

[3] Morris, Genevieve et al. "Patient Identification And Matching Final Report". HealthIT.gov. N.p., 2014. Web. 17 Sept. 2015.

[4] Zhu, Ying, et al. "When to conduct probabilistic linkage vs. deterministic linkage? A simulation study." Journal of Biomedical Informatics 56.C (2015): 80-86.

[5] Just, B. H., et al. "Managing the integrity of patient identity in health information exchange." Journal of AHIMA/American Health Information Management Association 80.7 (2009): 62-69.

[6] Hogan, William R., and Michael M. Wagner. "Accuracy of data in computer-based patient records." Journal of the American Medical Informatics Association 4.5 (1997): 342-355. institutions are only able to record minimal identifying information about

[7] Bell, Glenn B., and Anil Sethi. "Matching records in a national medical patient index." Communications of the ACM 44.9 (2001): 83-88.

[8] Gill, Leicester. "OX-LINK: the Oxford medical record linkage system." (1997).

The approach taken here is influenced in large part by the methods of OX-LINK. Subsequent improvements to such probabilistic techniques have been incorporated, as well.

2.2 Phonemic Tokenization

Phonemic name compression, indexing, or tokenization schemes use phonetics to approximately represent a word or name. There are several common name compression schemes in wide use, including Soundex, NYSIIS, metaphone, and double metaphone, which appear here in chronological order according to their date of creation [9] [10]. The purpose of name compression in record linkage is to allow for a potential name match when the spelling of two names disagree but the phonetics are identical. For example, the Soundex code for Catie and Caity are both **C300**, although their spelling is different.

Soundex is the oldest method here, developed in the early 1900s and used to aid the U.S.A. Census Bureau [11]. It is computationally efficient and included in several modern databases for fuzzy name matching for that reason, but its shortcomings are quite obvious when non-Anglo-Saxan names are used and in other scenarios. Continuing the example in the previous paragraph, the Soundex code for Katie is **K300**, although it sounds identical to both Catie and Caity, which both have the code **C300**. After stripping vowels and other characters in certain situations, Soundex only looks at the initial part of a name.

NYSIIS was developed in the 1970s and is used by the New York State Department of Health and Criminal Justice Services. Unlike Soundex, vowels are not dropped and codes are not truncated to just four characters. For example, the NYSIIS encoding of Jonathan is **JANATAN**. This characteristic leads to improvements in a number of areas, and the algorithm is purported to better handle phonemes that occur in Hispanic and some European names. The NYSIIS codes for Catie, Caity, and Katie are all **CATY**. The improvement can be seen here since NYSIIS correctly identifies the same code for these phonetically identical names.

Metaphone, and then double metaphone, are the most recent phonemic compressions available in EHRcorral [12] [13]. Metaphone was first published in 1990 and is the first algorithm here to consider the sequences of letters and sounds rather than just individual characters. It also performs its compression based on the entire name, not a truncated or stripped version. Double metaphone was released ten years after metaphone, and particularly turns its attention toward accounting for combinations of sounds that are not present in the english language. This makes double metaphone suitable for compression of english or anglicized names of a variety of origins, including Chinese, European, Spanish, Greek, French, Italian, and more. It is the most robust algorithm not only for that reason, but also because it produces two encodings per name: a primary encoding and a secondary encoding. The metaphone codes for Catie, Caity, and Katie are all **KT**. Double metaphone produces just one encoding (again, **KT**) and drops the secondary encoding since this is a phonetically simple name. If we consider the name Katherine, metaphone produces **KORN** while double metaphone generates two encodings, **KORN, KTRN**.

Phonemic compressions have been widely used to quickly identify similar names for record linkage. They can quickly identify similar names and exclude dissimilar ones, reducing the time to find matches, and they can improve false positive/negative rates by eliminating unnecessary matches. They are important to understand in the context of *Record Blocking*.

[9] Alvey, W., and B. Jamerson. "Record Linkage Techniques—1997: Proceedings of an International Workshop and Exposition." Washington, DC: Federal Committee on Statistical Methodology (1997).

[10] Dolby, James L. "An algorithm for variable-length proper-name compression." Information Technology and Libraries 3.4 (2013): 257-275.

[11] Beider, Alexander, and Stephen Morse. "Phonetic Matching: A Better Soundex". http://stevemorse.org. N.p., 2015. Web. 17 Oct. 2015.

[12] Lawrence, Philips. "Hanging on the metaphone." Computer Language 7.12 (1990): 39-43.

[13] Philips, Lawrence. "The double metaphone search algorithm." C/C++ users journal 18.6 (2000): 38-43.

2.3 Record Blocking

Record blocking is a technique used to eliminate probabilistic matching between records that clearly do not match based on some field, such as last name [14] [15]. If every record has to be checked against every other record for a probabilistic match there are $\binom{n}{2}$ checks that must occur. For n=1,000,000 records, this would require 499,999,500,000 (499 trillion) record-to-record comparisons. If every comparison takes just 1 microsecond, it would still take over 5 days for the matching process to complete. However, if we were able to limit record-to-record comparisons to groups (i.e. blocks) of records that have the possibility of matching and ignore other record-to-record combinations, the time to completion could be greatly reduced.

By default, EHRcorral blocks data into groups by the phonemic compression of the current surname plus the first initial of the forename. Other blocking techniques group by phonemic compression of the forename or current surname, or by birth month or year. A combinatory approach can be taken, as well, blocking by both current surname and birth year, and then by sex and birth month. By probabilistically checking only records in the same block, the time until the algorithm finishes is greatly reduced if the average block size is manageable. Blocking by phonemic compression has the advantage of eliminating checks between two names that have similar spelling but different pronunciations, potentially eliminating false positives that might match based on word-distance measures alone. On the other hand, if the phonemic compression algorithm is inaccurate (as we saw with Caity and Katie using Soundex), potential matches are discarded, increasing the false negative rate.

Soundex, NYSIIS, and metaphone all generate a single encoding, while the more robust double metaphone generates two encodings. In the case of double metaphone both encodings are used, effectively creating larger block sizes. This can lead to a significant increase in computation time, depending on the data set. Therefore, the first initial of the forename is also used to then decrease the block size. This also helps reduce the size of blocks for very common surnames, such as Smith, which occurs at a rate of about 1% (or 10,000 for every one million) in the United States of America.

2.4 Exploding Data

Exploding the data set refers to the process of generating additional Records from each Record by combining, switching, or expanding fields. The purpose of exploding the data set is to mitigate the effect of certain data entry errors or scenarios encountered in EHRs, such as the transposition of first name and middle name, or the entry of a nickname in a name field. This process is used in conjunction with blocking in order to increase the potential matches of a record that might have these errors [8].

Consider a Record for a man named Bill Taft Robinson:

Forename: Bill
Mid-forename: Taft
Current surname: Robinson

Initially, blocking would be performed by taking the phonemic compression of the current surname plus the first initial of the forename. The primary double metaphone compression of Robinson is **RPNSN**, and adding on the first initial of the forename would put this record in block **RPNSNB**. When this record is exploded, it will get the following additional blocking groups:

- **RPNSNT**, using the first initial of the mid-forename
- **RPNSNW**, using William in place of Bill for the forename since Bill is a common nickname for William in the english language.

[14] Kelley, Robert Patrick. Blocking considerations for record linkage under conditions of uncertainty. Bureau of the Census, 1984.
[15] Clark, D. E. "Practical introduction to record linkage for injury research." Injury Prevention 10.3 (2004): 186-191.

This makes this Record available for probabilistic matching within three blocking groups. Therefore, if Bill Taft Robinson has another Record under William Taft Robinson, a potential match can be found with this Record. Note that the blocking group is only used to determine which Records are checked. It does not modify the forename, nor does it insert William in place of Bill.

A standard set of names and their nicknames is not yet included with EHRcorral, but in the future one can be supplied to customize the explosion to names from a different region. For example, instead of Bill and William, when dealing with records containing Hispanic and Western European names perhaps the European name Elizabeth should also be considered as Isabel, the accepted Spanish version of Elizabeth, for blocking purposes.

2.5 Matching

The matching that EHRcorral does is heavily based on the Oxford Record Linkage System (OX-Link) [8]. It takes a number of name and non-name fields and determines the similarities between two respective records. Based on the similarity weight calculated for each individual field, an aggregate similarity for the two records is determined.

EHRcorral cycles through every record to build a square symmetric similarity matrix. Thus, the similarity between any two records can be determined by looking at the matrix. By thresholding the similarity matrix, one can create a link between records with similarities above the threshold.

2.5.1 Similarity Measures

EHRcorral separates record similarity into two sections: name fields and non-name fields. Name fields alone have a high degree of accuracy in determining the similarity of two records [16] [17]. Thus, EHRcorral heavily weights matching based on names and uses the non-name fields for fine-tuning.

However, there are many types of entry errors [18].

- **character insertion**: Richard ⇒ Ricthard
- **character omission**: Sullivan ⇒ Sulivan
- **character substitution**: Robert ⇒ Rodert
- **character transposition**: 55414 ⇒ 55441
- **gender misclassification**: M ⇒ F

To deal with the first four errors, EHRcorral converts all characters to lowercase and uses the damerau-levenshtein edit distance measurement on most of its data fields [7]. Thus, if any of those errors occur, the similarity between the two fields compared is still high. To avoid the issue of gender misclassification as best as possible, EHRcorral focuses on sex in comparisons. Further work may be done in this area to handle better gender misclassification in the future. Birth date and postal code are converted to character fields to handle all of the character errors above and better understand the similarity of the fields between records.

The name fields have complex similarity calculations. These fields have the potential for a different type of transposition error than other fields. One may enter a forename as a mid-forename or vice versa. This can happen with current and birth surname as well. To account for this, EHRcorral checks both forename or surname fields in the second record when comparing it with the respective field from the first and takes the one with the highest similarity. This has the benefit of handling the case where a surname is changed, e.g. in marriage, much better. Once the similarity is determined, EHRcorral checks whether a given surname compression (see *Phonemic Tokenization* for compression

[16] Aldridge, Robert W., et al. "Accuracy of Probabilistic Linkage Using the Enhanced Matching System for Public Health and Epidemiological Studies." PloS one 10.8 (2015): e0136179.

[17] Weber, Susan C., et al. "A simple heuristic for blindfolded record linkage." Journal of the American Medical Informatics Association 19.e1 (2012): e157-e161.

[18] Theera-Ampornpunt, Nawanan, Boonchai Kijsanayotin, and Stuart M. Speedie. "Creating a large database test bed with typographical errors for record linkage evaluation." AMIA... Annual Symposium proceedings/AMIA Symposium. AMIA Symposium. 2007.

details) is common or rare or checks whether a given forename first letter is common or rare. The compression is used with surnames to negate potentially unique entry errors impacting the determination. The forename is less significant in determining the similarity of two records, so using just the first letter saves time computationally and avoids most entry errors while remaining relatively accurate. With the determination of a name being common or rare, the similarity is scaled accordingly and a weight is assigned, which can go negative since very dissimilar names should lead records to be considered very dissimilar.

The address field requires a lot cleaning before a weight is calculated. First, both address fields are combined and put into lowercase. Then, all abbreviations for address suffixes (e.g. avenue) and designators (e .g. apartment) are found and standardized based on the abbreviations that the United States Postal Service uses [19]. After this, the first 12 characters of the address are compared as mentioned above to account for the different types of character entry errors. Address fields that only have a couple entry errors still have some similarity weight, but ones that have more differences are given zero weight. This accounts for people moving around without diminishing the similarity too much.

The comparison of the respective postal code and national identification fields are relatively simple. EHRcorral looks for exact matches and single differences in determining similarity for these fields. Here, outside of simple entry errors, any field that is not exactly the same is considered no match at all. This is due to the fact that similar values for these fields are only meaningful in as much as they represent entry errors. Like with address, there are no negative weights for the postal code due to the potential for moving. National identifications do not have negative weights because of the difficulty with getting consistent entry in this area.

The similarity between two sex fields is very simple. EHRcorral asks for single character sex identification. If they are the same, a small positive weight is returned. If they are not, then a large negative weight is returned. This is due to the fact that a different sex should render two records significantly less similar, but the same sex means very little for their similarity.

The date of birth field has a slightly more complex comparison. The year, month, and day are each compared separately using the damerau-levenshtein method of calculating edit distance to account for all of the character errors mentioned above. Then, the total similarity is summed with extra weight given to the year, since entry errors are less likely there (i.e. someone is more likely to recognize that 1972 was keyed in as 9172), and different generations will be reflected in this area to separate family members with common birth days. This field has a strong influence amongst the non-name fields since it should never change and matches do imply that records are quite similar. Like with sex, there is a strong negative weight for records that are strongly dissimilar, but there is also a strong positive weight for the reasons mentioned above.

The summing of the weights is relatively simple once all individual weights are calculated. An algebraic sum is divided by the total possible weight that a record could have (this will vary based on commonality of forenames and surnames). This returns a values between zero and one that determines the probability that two records are the same. Then, thresholding can be applied to make actual determinations.

2.5.2 Similarity Matrix

The similarity matrix is calculated by using the record similarity function described above. As EHRcorral cycles through each record, it looks at the respective blocks for that record (see *Record Blocking* for details) and determines similarities for each record within the respective blocks. Then, the accession number for each record is used to fill in the correct row with the similarities in the correct columns. All records that are not in the same block as the one being compared receive a zero similarity score. The similarity of any two records can be found by looking up their respective accession numbers and then look at either row and column combination.

Thresholding can be used to determine the linkage of records. EHRcorral leaves to the user the determination of which threshold is appropriate based on the particular data set on which they are using EHRcorral.

[19] United States Postal Service. “Appendix C”. Pe.usps.gov. N.p., 2015. Web. 4 Dec. 2015.

References

CHAPTER 3

Installation

At the command line:

```
$ easy_install ehrcorral
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv ehrcorral
$ pip install ehrcorral
```

CHAPTER 4

Usage

To use EHRcorral in a project:

```
import ehrcorral
```

EHRCorral operates on a collection of Records, each of which represents a single electronic health record. A collection of Records is called a Herd, hence the name EHRCorral: generating a master patient index of all the records is done by "corralling" the Herd.

There is a small number of actions to perform, but potentially several setting to consider:

1. Create Records
2. Create a Herd
3. Populate the Herd with the Records
4. Corral the Herd

4.1 Records

Ref: *ehrcorral.ehrcorral.Record*

A Record is a simplified representation of a patient's EHR which only contains information relevant to the current matching algorithm. Each Record *must* contain a forename and a current surname, but it can also house other identifying information. All the information in a Record is used to discover other Records that describe the same individual.

```
ehr_entries = [
    {
        'forename': 'John',
        'mid_forename': '',
        'current_surname': 'Doe',
        'suffix': 'Sr.',
        'address1': '1 Denny Way',
        'city': 'Orlando',
        'state_province': 'FL'
        'postal_code': 32801,
        'sex': 'M',
        'gender': 'M',
        'national_id1': '123-45-678',  # Using field as social security num
        'id2': 'F1234578',  # Optional ID, such as driver license num
        'birth_year': '1985',
        'birth_month': '08',
        'birth_day': '04',
```

```
        'blood_type': 'B+'
    },
    {
        'first_name': 'Jane',
        'middle_name': 'Erin',
        'birth_surname': 'Doe',
        'current_surname': 'Fonda',
        'suffix': '',
        'address1': '1 Bipinbop St',
        'address2': 'Apt. 100',
        'city': 'Austin',
        'state_province': 'TX',
        'postal_code': 73301,
        'sex': 'F',
        'gender': 'F',
        'national_id1': '876-54-321',
        'birth_year': 1976,
        'birth_month': '08',  # Numeric fields are coerced to proper type
        'birth_day': 01,
        'blood_type': 'A-',
    }
]
records = [ehrcorral.gen_record(entry) for entry in ehr_entries]
```

Above, we create two Records (an entry for John and one for Jane) using the function *ehrcorral.ehrcorral.gen_record()*. Generally, you will not need to interact directly with the Records once they are created.

In practicality, you won't have just two EHR entries, but thousands or millions of them, and there might be multiple entries for John or Jane and many other individuals in the sub-population. The Record class is designed to be extremely light on memory usage, much more so than a dictionary or list, for example. A collection of 10 million Records will occupy about 5—6 GB, whereas 10 million dictionaries containing the same data will occupy about three times the memory. Therefore, when generating Records it is advisable *not* to build up a large dictionary of data to then be sent one by one to *ehrcorral.ehrcorral.gen_record()*. Instead, generate the Records in a loop that operates only on a single EHR entry at a time so the dictionaries like the ones above are thrown away once the Record is created:

```
records = []
for entries in ehr:
    # Extract forenames, sex, etc. from EHR data into dict called 'entry'
    # ...
    # entry =  {'forename': 'John', ... , 'blood_type': 'B+'}
    records.append(ehrcorral.gen_record(entry))
```

4.2 Record Fields

For the full list of fields available to generate a Record, see *ehrcorral.ehrcorral.Profile*.

If additional fields are passed to `gen_record()` they are ignored. Missing fields receive a value of empty string. No transformations are applied to these fields other than to coerce strings to integers when the algorithm requires integers. You should perform any pre-processing that you think is relevant for your region or data set, such as removing accents or umlauts if you do not want to match based on such special characters, defining forename and mid forename if names in your region are particularly long, removing prefixes like Mr. and Mrs., and determining what to use for the national ID field.

4.3 Creating a Herd

Ref: *ehrcorral.ehrcorral.Herd.populate()*

Once the Records have been created, you can populate a Herd. A list or tuple of Records can be used.

```
herd = ehrcorral.Herd()
herd.populate(records)
```

In order to prevent race conditions during matching, the population of a Herd cannot be updated once it is set. Calling `populate()` again with additional records will raise an error.

4.4 Matching Records

Ref: *ehrcorral.ehrcorral.Herd*

To performing record-linkage on the Herd, you call its `corral()` method. This method requires as input a function which performs phonemic name compression, for Record blocking purposes. For convenience, Soundex, NYSIIS, metaphone, and double metaphone implementations have been included. Below, double metaphone is used. If you are not yet familiar with blocking methods, please consult *Record Blocking* in the documentation.

```
from ehrcorral.compressions import dmetaphone
# Alternate blocking compressions:
# from ehrcorral.compressions import soundex
# from ehrcorral.compressions import nysiis
# from ehrcorral.compressions import metaphone
# from ehrcorral.compressions import first_letter
herd.corral(blocking_compression=dmetaphone)
similarities = herd.similarity_matrix
```

See *ehrcorral.ehrcorral.Herd.corral()* for documentation of additional function parameters.

Running `corral()` on the Herd generates a similarity (i.e. probability) matrix with dimension N _x_ N, where N is the number of records in the Herd. This matrix provides the probabilities that each record belongs to the same person as contained in every other record in the Herd. Each row and column index in the similarity matrix corresponds to each Record's `record_number` property (see documentation for Record class). The user can decide how to link records using a threshold value to determine which records belong to the same individual. Currently there is no built-in method to automatically merge records together since there are many different strategies for merging that the user might want to employ. Additionally, it is likely that the user would want to merge the original data that was used to generate each Record rather than merging the Records themselves.

CHAPTER 5

Modules

5.1 ehrcorral.ehrcorral

5.1.1 Herd

class ehrcorral.ehrcorral.**Herd**

A collection of *Record* with methods for interacting with and linking records in the herd.

similarity_matrix

numpy.ndarray, None

A numpy array containing the similarities between *Record* instances, ordered by accession number on both axes. Each entry is between 0 and 1 with 1 being perfect similarity.

append_block_dict(*record*)

Appends the herd's block dictionary with the given Record's blocking codes.

The dictionary keys are block codes. The value of each key is a list of references to Records that have that block.

Parameters **record** (*Record*) – An object of class *Record*

append_names_freq_counters(*record*)

Adds the forename and surname for the given Record to the forename and surname counters.

Parameters **record** (*Record*) – An object of class *Record*

corral(*forename_freq_method=<function first_letter>*, *surname_freq_method=<function doublemetaphone>*, *blocking_compression=<function doublemetaphone>*)

Perform record matching on all Records in the Herd.

Parameters

- **forename_freq_method** (*func*) – A function that performs some sort of compression. Compression of forename can be different than compression of surname. The compression information is used to determine weights for certain matching scenarios. For example, if forename is compressed to be just the first initial, matching a name that begins with the letter 'F' will result in a weight equal to the fraction of names that begin with the letter 'F' in the entire Herd. The less common names that begin with 'F' are, the more significant a match between two same or similar forenames that begin with 'F' will be. Defaults to the first initial of the forename.
- **surname_freq_method** (*func*) – A function that performs some sort of compression. Defaults to double metaphone.

- **blocking_compression** (*func*) – Compression method to use when blocking. Blocks are created by compressing the surname and then appending the first initial of the forename. Defaults to double metaphone and then uses the primary compression from that compression. By default the first initial of the forenames are appended to the surname compressions to generate block codes.

populate(*records*)
Sets the Herd's sub-population.

Parameters **records** (*list, tuple*) – A list or tuple containing multiple *Record*

size
Returns the size of the Herd's population.

5.1.2 Record

class ehrcorral.ehrcorral.**Record**
A Record contains identifying information about a patient, as well as generated phonemic and meta information.

gen_blocks(*compression*)
Generate and set the blocking codes for a given record.

Blocking codes are comprised of the phonemic compressions of the profile surnames combined with the first letter of each forename. Generated blocking codes are stored in self._blocks, and only contain the unique set of blocking codes.

Parameters **compression** (*func*) – A function that performs phonemic compression.

save_name_freq_refs(*record_number*, *forename_freq_method*, *surname_freq_method*)
Compress the forenames and surnames and save the compressions to the Record.

Parameters

- **record_number** (*int*) – An integer to be assigned as initial person and accession number.
- **forename_freq_method** (*func*) – A function that performs some sort of compression on a single name.
- **surname_freq_method** (*func*) – A function that performs some sort of
- **on a single name.** (*compression*) –

5.1.3 Profile

class ehrcorral.ehrcorral.**Profile**
A selection of patient-identifying information from a single electronic health record.

All fields should be populated with an int or string and will be coerced to the proper type for that field automatically.

forename
Also known as first name.

mid_forename
Also known as middle name.

birth_surname
Last name at birth, often same as mother's maiden name.

current_surname
Current last name. Can differ from birth surname often in the case of marriage for females.

suffix
Sr., Junior, II, etc.

address1
Street address, such as "100 Main Street".

address2
Apartment or unit information, such as "Apt. 201".

state_province
State or province.

postal_code

country
Consistent formatting should be used. Do not use USA in one Record and United States of America in another.

sex
Physiological sex (M or F)

gender
The gender the patient identifies with (M or F), e.g. in the case of transexualism.

national_id1
For example, social security number. This should be the same type of number for all patients. Do not use USA social security in one Record and with Mexico passport number in another.

id2
Can be used as an additional identifying ID number, such as driver's license number. Again, define the type of ID number this is for the entire sub-population.

mrn
Medical record number.

birth_year
In the format YYYY.

birth_month
In the format MM.

birth_day
In the format DD.

blood_type
One of A, B, AB, or O with an optional +/- denoting RhD status.

5.1.4 gen_record()

ehrcorral.ehrcorral.**gen_record**(*data*)
Generate a *Record* which can be used to populate a Herd.

In addition to extracting the profile information for

Parameters **data** (*dict*) – A dictionary containing at least one of fields in PROFILE_FIELDS.

Returns py:class:.*Record*.

Return type A object of class

5.1.5 compress()

ehrcorral.ehrcorral.**compress**(*names*, *method*)
Compresses surnames using different phonemic algorithms.

Parameters

- **names** (*list*) – A list of names, typically surnames
- **method** (*func*) – A function that performs phonemic compression

Returns A list of the compressions.

5.2 ehrcorral.measures

5.2.1 record_similarity()

ehrcorral.measures.**record_similarity**(*herd*, *first_record*, *second_record*, *forename_method=<function damerau_levenshtein>*, *surname_method=<function damerau_levenshtein>*)
Determine weights for the likelihood of two records being the same.

Parameters

- **herd** (Herd) – An object of *Herd* which contains the two records being compared.
- **first_record** (Record) – An object of *Record* to be compared to the other one.
- **second_record** (Record) – An object of *Record* to be compared to the other one.
- **forename_method** (*func*) – A function that performs some sort of comparison between strings.
- **surname_method** (*func*) – A function that performs some sort of comparison between strings.

Returns A tuple of the sum of name weights and the sum of non-name weights.

5.2.2 get_forename_similarity()

ehrcorral.measures.**get_forename_similarity**(*herd*, *records*, *method*, *name_type*)
Determine weights for the likelihood of two forenames being the same.

Parameters

- **herd** (Herd) – An object of *Herd* which contains the two records being compared.
- **records** (*List[Record]*) – A list of two objects of *Record* to be compared to one another.
- **method** (*func*) – A function to be used to compare the forenames.
- **name_type** (*unicode*) – A unicode string to indicate which forename is being compared.

Returns The forename weight for the similarity of the forenames.

5.2.3 extract_forename_similarity_info()

ehrcorral.measures.**extract_forename_similarity_info**(*herd*, *record*, *name_type*)
Extract desired forename and associated frequency weight.

Parameters

- **herd** (Herd) – An object of *Herd* which contains the frequency dictionary used for the frequency weight.
- **record** (Record) – An object of *Record* from which to extract the forename.
- **name_type** (*unicode*) – A unicode string to indicate which forename is being extracted.

Returns The forename and associated frequency weight for requested name.

5.2.4 get_surname_similarity()

ehrcorral.measures.**get_surname_similarity**(*herd*, *records*, *method*, *name_type*)
Determine weights for the likelihood of two surnames being the same.

Parameters

- **herd** (Herd) – An object of *Herd* which contains the two records being compared.
- **records** (*List[Record]*) – A list of two objects of *Record* to be compared to one another.
- **method** (*func*) – A function to be used to compare the surnames.
- **name_type** (*unicode*) – A unicode string to indicate which surname is being compared.

Returns The surname weight for the similarity of the surnames.

5.2.5 extract_surname_similarity_info()

ehrcorral.measures.**extract_surname_similarity_info**(*herd*, *record*, *name_type*)
Extract desired surname and associated frequency weight.

Parameters

- **herd** (Herd) – An object of *Herd* which contains the frequency dictionary used for the frequency weight.
- **record** (Record) – An object of *Record* from which to extract the surname.
- **name_type** (*unicode*) – A unicode string to indicate which surname is being extracted.

Returns The forename and associated frequency weight for requested name.

5.2.6 get_address_similarity()

ehrcorral.measures.**get_address_similarity**(*records*, *method=<function damerau_levenshtein>*)
Determine weights for the likelihood of two addresses being the same.

Parameters

- **records** (*List[Record]*) – A list of two objects of *Record* to be compared to one another.
- **method** (*func*) – A function to be used to compare the addresses.

Returns The address weight for the similarity of the addresses.

5.2.7 clean_address()

ehrcorral.measures.**clean_address**(*address*)

Clean unicode string that contains an address of all punctuation and standardize all street suffixes and unit designators.

Parameters **address** (*unicode*) – A unicode string that contains an address to be cleaned and standardized.

Returns The cleaned unicode address string.

5.2.8 get_post_code_similarity()

ehrcorral.measures.**get_post_code_similarity**(*records*, *method=<function damerau_levenshtein>*)

Determine weights for the likelihood of two postal codes being the same.

Parameters

- **records** (*List[Record]*) – A list of two objects of *Record* to be compared to one another.
- **method** (*func*) – A function to be used to compare the postal codes.

Returns The postal code weight for the similarity of the postal codes.

5.2.9 get_sex_similarity()

ehrcorral.measures.**get_sex_similarity**(*records*)

Determine weights for the likelihood of two sexes being the same.

Parameters **records** (*List[Record]*) – A list of two objects of *Record* to be compared to one another.

Returns The sex weight for the similarity of the sexes.

5.2.10 get_dob_similarity()

ehrcorral.measures.**get_dob_similarity**(*records*, *method=<function damerau_levenshtein>*)

Determine weights for the likelihood of two dates of birth being the same.

Parameters

- **records** (*List[Record]*) – A list of two objects of *Record* to be compared to one another.
- **method** (*func*) – A function to be used to compare the dates of birth.

Returns The date of birth weight for the similarity of the dates of birth.

5.2.11 get_id_similarity()

ehrcorral.measures.**get_id_similarity**(*records*, *method=<function damerau_levenshtein>*)

Determine weights for the likelihood of two national IDs being the same.

Parameters

- **records** (*List[Record]*) – A list of two objects of *Record* to be compared to one another.
- **method** (*func*) – A function to be used to compare the national IDs.

Returns The national ID weight for the similarity of the two national IDs.

CHAPTER 6

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at https://github.com/nsh87/ehrcorral/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

6.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

6.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

6.1.4 Write Documentation

EHRCorral could always use more documentation, whether as part of the official EHRCorral docs, in docstrings, or even on the web in blog posts, articles, and such.

6.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/nsh87/ehrcorral/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

6.2 Get Started!

Ready to contribute? Here's how to set up *ehrcorral* for local development.

1. Fork the *ehrcorral repo on GitHub <https://github.com/nsh87/ehrcorral>*.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/ehrcorral.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv ehrcorral
$ cd ehrcorral/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 ehrcorral tests
$ pylint ehrcorral tests -f colorized
$ python setup.py test
$ tox
```

   To get flake8, pylint, and tox, just pip install them into your virtualenv. You can install all the recommended dependencies with:

```
$ pip install -r requirements_dev.txt
```

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.

3. The pull request should work for Python 2.6, 2.7, 3.3, and 3.4, and for PyPy. Check https://travis-ci.org/nsh87/ehrcorral/pull_requests and make sure that the tests pass for all supported Python versions.

6.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_ehrcorral
```

CHAPTER 7

Project Info

7.1 Release Log

7.1.1 0.0.1 (2015-12-17)

- **First release on PyPI.**
  - Docs need updating and more usage.
  - Corralling is functional, but does not use nickname expansions.
  - Some tweaking of probabilities could be attempted for certain scenarios.
  - Probabilities are generated using birth year, name fields, address, and zip code.

7.1.2 0.0.2 (2015-12-17)

- **Update documentation.**
  - Add significant documentation of matching algorithm in Overview docs.
  - Fix up code examples in Usage docs.

7.1.3 0.0.3 (2015-12-17)

- **Another update to documentation.**
  - The previous release had some minor docs stuff missing.

7.2 Authors

Nikhil Haas <nikhil@nikhilhaas.com>

7.3 Contributors

None yet. Why not be the first?

CHAPTER 8

Indices and tables

- genindex
- modindex
- search