ipprl_tools Documentation: Linkability Measures v1.0

July 2019

Contents

1 Compatibility

The *ypprl_tools* package was written using Python 3.6, but should be compatible with any version of Python 3 (Python 3.x).

2 Required Dependencies

The following packages are required dependencies for the *ipprl_tools* package. If you installed *ipprl_tools* through PIP, these dependencies should be installed automatically.

- **Pandas** $\geq$ v0.23
  - `https://pandas.pydata.org`
- **NumPy** $\geq$ v1.16

    - https://www.numpy.org

- **SciPy** $\geq$ v1.2
    - https://www.scipy.org

3 Optional Dependencies

The following packages are optional dependencies for the *ipprl_tools* package. These dependencies will not be installed automatically when installing *ipprl_tools* with PIP, so they must be installed manually if needed.

- **Fuzzy** $\geq$ v1.2.2
    - This package is required for the Soundex corruption method. For more information about the package, visit `https://pypi.org/project/Fuzzy/`.
- **Jupyter** $\geq$ v1.0.0
    - This package is required to view and run the tutorial Jupyter notebook. For more information about Jupyter, visit `https://jupyter.org/`

4 Installation

4.1 PIP Method (Recommended)

To install the package via PIP run the command:

```
pip install git+git://github.com/cu-recordlinkage/ipprl_tools
```

through a command-line interface.

This command will install the *ipprl_tools* package into your default Python environment. This command will also install the required dependencies (Pandas, NumPy, SciPy, etc) if they are not already installed.

4.2 GitHub Method

: The source code can also be cloned directly from GitHub using the following command from a command-line interface.

```
git clone https://github.com/cu-recordlinkage/ipprl_tools
```

5 Usage

5.1 Importing the Package

To use *ipprl_tools*, first import the `metrics` submodule.

```
In [ ]: from ipprl_tools import metrics
```

Figure 1: Importing *ipprl_tools*

This command will import all of the functions defined in the `metrics` submodule. If you only need a subset of the functions, you can specify the exact functions to import as well:

```
In [ ]: from ipprl_tools.metrics import convert_data,run_metrics
```

Figure 2: Importing specific functions

5.2 Data Prerequisites

The linkability metric functions expect that data will be contained in a Pandas DataFrame. To read in a file using Pandas, first call the appropriate read function. In our case we are reading CSV data, so we use *pandas.read_csv()*, but alternative functions are available for other types of data. For additional ways to import data using Pandas, refer to the Pandas Documentation here:

Pandas Documentation: IO

```
In [6]: import pandas as pd
        data = pd.read_csv("old/corrupted_v2.csv")
        data.head(5)
```

Out[6]:

	first_name	last_name	email	address
0	Isabelita	Dommersen	NaN	48 Grover Way
1	Caspar	Le Moucheux	NaN	158 Marquette Hill
2	Garwin	Ismirnioglou	gismirnioglou2@army.mil	9538 Lighthouse Bay Circle
3	Ewan	Paquet	NaN	5768 Kensington Street
4	Kamila	Tailour	ktailour4@rediff.com	8 Lindeanr Terrace

Figure 3: Reading in a CSV file with Pandas

The linkability metric functions provided in *ipprl_tools* are designed to operate on Pandas DataFrames, but in order for them to work correctly, we must first ensure that the DataFrame is in the correct format.

By default, Pandas will attempt to parse columns of your input file differently, depending on the type of data in the column.

```
In [6]:  data = pd.read_csv("corrupted_3_parts.csv")
          data.dtypes

Out[6]: id                   int64
        net_id               int64
        first_name          object
        last_name           object
        email               object
        address1            object
        ssn                 object
        gender              object
        city                object
        zip                float64
        state               object
        dob                 object
        phone               object
        phone2              object
        phone3              object
        race                object
        pcp_npi             object
        suffix              object
        title               object
        middle_name        float64
        mothers_maiden     float64
        address2           float64
        phone4             float64
        mrn                float64
        dtype: object
```

Figure 4: Data types of each column, after reading data from a CSV file.

In this case, Pandas has parsed some of the numerical columns (*id*, *zip*, etc.) as different data types. In order for the linkability metrics to work, we first need to convert all columns in the DataFrame to the *string* data type. We also need to ensure that missing data is handled in the correct way. All linkability metrics treat the empty string ("") as a missing value.

In order to make this process easy, *npprl_tools* contains a function called *convert_data()* (6.2.1), which can be used to automatically convert a DataFrame to the correct format. To use, import *convert_data* from the *npprl_tools.metrics* submodule, then call it on your DataFrame.

```
In [8]:  data = convert_data(data)
         data.dtypes

Out[8]:  id               object
         net_id           object
         first_name       object
         last_name        object
         email            object
         address1         object
         ssn              object
         gender           object
         city             object
         zip              object
         state            object
         dob              object
         phone            object
         phone2           object
         phone3           object
         race             object
         pcp_npi          object
         suffix           object
         title            object
         middle_name      object
         mothers_maiden   object
         address2         object
         phone4           object
         mrn              object
         dtype: object
```

Figure 5: Example of calling *convert_data()* and viewing the result.

In the above figure, we call *metrics.convert_data()* on our DataFrame, and view the data types of the output. We can see that the DataFrame was converted correctly, as every column is now of *type(object)*.

5.3 Computing Metrics

5.3.1 Calling Individual Functions

To call a linkability metric on a DataFrame, first ensure that you've imported the function by following the instructions in Section 5.1.

We can call an individual function on every column in the DataFrame, or specify some subset of the columns to use.

```
In [10]:  metrics.theoretical_maximum_entropy(data,columns=["first_name","last_name"])

Out[10]: {'first_name': 14.306488964728633, 'last_name': 15.712123926676279}
```

Figure 6: Example of calling a linkability metric by manually specifying the columns to operate on.

```
In [11]: metrics.theoretical_maximum_entropy(data)

Out[11]: {'id': 17.042599881712917,
          'net_id': 16.609640474436812,
          'first_name': 14.306488964728633,
          'last_name': 15.712123926676279,
          'email': 14.965919530982598,
          'address1': 16.66379434619952,
          'ssn': 16.38407544713785,
          'gender': 1.5849625007211563,
          'city': 13.272775563286087,
          'zip': 13.45455634396194,
          'state': 10.521600439723727,
          'dob': 12.467350814487581,
          'phone': 14.253847484987404,
          'phone2': 14.261507309202056,
          'phone3': 16.49039651667306,
          'race': 11.326991174900817,
          'pcp_npi': 10.396604781181859,
          'suffix': 2.584962500721156,
          'title': 11.88683970588442,
          'middle_name': -0.0,
          'mothers_maiden': -0.0,
          'address2': -0.0,
          'phone4': -0.0,
          'mrn': -0.0}
```

Figure 7: Example of calling a linkability metric by using the default column argument (all columns)

In either case, the output of each linkability metric is a Python *dict* object, which has keys corresponding to the names of the selected columns, and values corresponding to the calculated linkability metric.

5.3.2 Using *run_metrics()*

Although each metric can be run individually, *ipprl_tools* also provides a helper function to run all of the metrics at once, and provide a formatted output DataFrame.
To use this function, import *run_metrics* from the *ipprl_tools.metrics* submodule, and call it on your DataFrame.

```
In [16]: from ipprl_tools.metrics import run_metrics
         run_metrics(data)

Out[16]:
```

	Mean Group Size	Median Group Size	Stdev Group Size	Min Group Size	Max Group Size	Missing Data Ratio	Distinct Values Ratio	Shannon Entropy	Theoretical Max Entropy	% Theoretical Max Entropy	Average Token Frequency
id	1.000000	1.0	0.000000	1	1	0.00	1.000000	17.042600	17.042600	100.000000	1.000000
net_id	1.350000	1.0	0.653835	1	3	0.00	0.740741	16.468164	16.609640	99.148225	1.350000
first_name	6.463156	1.0	7.368322	1	51	0.03	0.150089	13.228706	14.306489	92.466477	6.663047
last_name	2.439456	2.0	1.766160	1	20	0.03	0.397637	15.118881	15.712124	96.224301	2.514903
email	1.054622	1.0	0.238512	1	3	0.75	0.237059	4.545562	14.965920	30.372755	4.218486
address1	1.222261	1.0	0.524363	1	7	0.06	0.769074	15.892898	16.663794	95.373825	1.300277
ssn	1.183864	1.0	0.458786	1	3	0.25	0.633526	13.034538	16.384075	79.556139	1.578486

Figure 8: Example of importing and using the *run_metrics* function.

The output of the function is a formatted Pandas DataFrame, where each column corresponds to a linkability metrics, and the rows are the columns from your original DataFrame.

6 Function Documentation

6.1 Linkability Metrics

6.1.1 *missing_data_ratio(data, columns=None)*

This function calculates the Missing Data Ratio, or MDR for each of the specified columns. The Missing Data Ratio is defined as:

$$MDR_i = \frac{Number\,of\,records\,with\,missing\,value\,in\,Variable_i}{Total\,number\,of\,records\,in\,Variable_i} \tag{1}$$

This metric can be used to determine what fraction of the columns values are usable for linkage.

Parameters:

- *data* - The Pandas DataFrame to be modified. Please ensure that your data is in the correct format (as specified in Section 5.2) before passing as an argument.
- *columns* - An optional list of columns to operate on. These columns should correspond to column names in the DataFrame that you would like to calculate metrics for. The default argument (*columns=None*) will calculate metrics for each column in *data*.

Return Value:

- *values* - A dictionary containing a key/value mapping of *column_name* → *mdr_value* for each column specified in *columns*.

6.1.2 *distinct_values_ratio(data, columns=None)*

This function calculates the Distinct Values Ratio (DVR), which is defined as:

$$DVR_i = \frac{Number\,of\,distinct\,values\,for\,Variable_i}{Number\,of\,records\,in\,Variable_i} \tag{2}$$

This metric can be used to determine how many unique values a variable/column has, relative to its size. A high DVR indicates that there are a large number of distinct values in the column, whereas a low DVR indicates that there are fewer distinct values.

Parameters:

- *data* - The Pandas DataFrame to be modified. Please ensure that your data is in the correct format (as specified in Section 5.2) before passing as an argument.
- *columns* - An optional list of columns to operate on. These columns should correspond to column names in the DataFrame that you would like to calculate metrics for. The default argument (*columns=None*) will calculate metrics for each column in *data*.

Return Value:

- *values* - A dictionary containing a key/value mapping of *column_name* → *dvr_value* for each column specified in *columns*.

6.1.3 *group_size(data, columns=None)*

This function calculates and returns the Group Sizes for each distinct value in each *column*. Group Size is defined as:

$$GS_{i,j} = Number\, of\, rows\, for\, Variable_i, Value_j \tag{3}$$

This function is used to calculate *agg_group_size* (6.1.4), but may also be called by the user.
Parameters:

- *data* - The Pandas DataFrame to be modified. Please ensure that your data is in the correct format (as specified in Section 5.2) before passing as an argument.
- *columns* - An optional list of columns to operate on. These columns should correspond to column names in the DataFrame that you would like to calculate metrics for. The default argument (*columns=None*) will calculate metrics for each column in *data*.

Return Value:

- *values (dict)* - For each column name in *columns*, returns a Counter object holding key/value mappings of *distinct_value* → *num_of_occurrences*

6.1.4 *agg_group_size(data, agg_func = np.mean, columns=None)*

This function calculates the aggregate group size for each column specified in *columns*, based on an arbitrary function *agg_func*. The function calls *group_size()* (6.1.3), and uses *agg_func* to reduce the results to a scalar value for each column.
Parameters:

- *data* - The Pandas DataFrame to be modified. Please ensure that your data is in the correct format (as specified in Section 5.2) before passing as an argument.
- *agg_func* - This is an arbitrary reduction function which accepts a list of values, and returns a scalar. Some examples of compatible reduction/aggregation functions are: *numpy.amin*, *numpy.amax*, *numpy.mean*, and *numpy.std*. The default argument (*agg_func = np.mean*) will pass NumPy's mean function to compute the mean group size for each column name specified in *columns*.
- *columns* - An optional list of columns to operate on. These columns should correspond to column names in the DataFrame that you would like to calculate metrics for. The default argument (*columns=None*) will calculate metrics for each column in *data*.

Return Value:

- *values* - A dictionary containing a key/value mapping of *column_name* → *agg_group_size_val* for each column specified in *columns*.

6.1.5 *shannon_entropy(data, columns=None)*

This function calculates the Shannon Entropy for each column name specified in *columns*. Shannon Entropy is defined as:

$$H_x = -\sum_{i=1}^{N} P(i)\, log_2\, P(i) \tag{4}$$

This metric is useful for determining the amount of information containing in a column. Typically, columns with large numbers of evenly distributed unique values will have higher values for Shannon Entropy.

Parameters:

- *data* - The Pandas DataFrame to be modified. Please ensure that your data is in the correct format (as specified in Section 5.2) before passing as an argument.
- *columns* - An optional list of columns to operate on. These columns should correspond to column names in the DataFrame that you would like to calculate metrics for. The default argument (*columns=None*) will calculate metrics for each column in *data*.

Return Value:

- *values* - A dictionary containing a key/value mapping of *column_name* → *shannon_entropy* for each column specified in *columns*.

6.1.6 *theoretical_maximum_entropy(data, columns=None)*

This function calculates the Theoretical Maximum Entropy (TME) for a column, given its current number of distinct values. The formal definition for Theoretical Maximum Entropy is:

$$TME_x = -log_2(\frac{1}{N}) \tag{5}$$

where N is the number of distinct values in the column.

Parameters:

- *data* - The Pandas DataFrame to be modified. Please ensure that your data is in the correct format (as specified in Section 5.2) before passing as an argument.
- *columns* - An optional list of columns to operate on. These columns should correspond to column names in the DataFrame that you would like to calculate metrics for. The default argument (*columns=None*) will calculate metrics for each column in *data*.

Return Value:

- *values* - A dictionary containing a key/value mapping of *column_name* → *theo_max_entropy* for each column specified in *columns*.

6.1.7 *percent_theoretical_maximum_entropy(data, columns=None)*

This function will calculate the Percent of Theoretical Maximum Entropy (PTME) for each column name specified in *columns*. The Percent of Theoretical Maximum Entropy is defined as:

$$PTME_x = \frac{H_x}{TME_x} * 100 \tag{6}$$

Where H_x and TME_x are the Shannon Entropy and Theoretical Maximum Entropy of the column, respectively. This function will call *theoretical_maximum_entropy()* (6.1.5) and *percent_theoretical_maximum_entropy* (6.1.6) in order to calculate the result.

Parameters:

- *data* - The Pandas DataFrame to be modified. Please ensure that your data is in the correct format (as specified in Section 5.2) before passing as an argument.
- *columns* - An optional list of columns to operate on. These columns should correspond to column names in the DataFrame that you would like to calculate metrics for. The default argument (*columns=None*) will calculate metrics for each column in *data*.

Return Value:

- *values* - A dictionary containing a key/value mapping of *column_name* $\rightarrow$ *pct_theo_max_entropy* for each column specified in *columns*.

6.1.8 *average_token_frequency(data, columns=None)*

This function will calculate the Average Token Frequency (ATF) for each column name specified in *columns*. The Average Token Frequency is defined as:

$$ATF = \frac{|V|}{N} \tag{7}$$

Where $|V|$ is the size of the column, and N is the number of unique values in the column.

Parameters:

- *data* - The Pandas DataFrame to be modified. Please ensure that your data is in the correct format (as specified in Section 5.2) before passing as an argument.
- *columns* - An optional list of columns to operate on. These columns should correspond to column names in the DataFrame that you would like to calculate metrics for. The default argument (*columns=None*) will calculate metrics for each column in *data*.

Return Value:

- *values* - A dictionary containing a key/value mapping of *column_name* $\rightarrow$ *avg_tok_freq* for each column specified in *columns*.

6.2 Utilities

The utilities functions are a set of functions provided alongside the Linkability Metric functions in order to perform common tasks associated with calculating linkability metrics on a dataset.

6.2.1 *convert_data(data)*

This function will convert the Pandas DataFrame *data* into a format suitable for calculating linkability metrics. The function will convert *data* to a DataFrame of string objects, automatically converting NaN values to the empty string ("").

Parameters:

- *data* - The Pandas DataFrame to be converted. If your DataFrame contains columns that are not of *type(str)*, or a value for NaN that is not the empty string (""), you will need to call this function to convert your DataFrame before running any of the linkage metrics.

Return Value:

- *data* - A version of the input DataFrame that has been converted to *type(str)*, and had all NaN values replaced with the empty string.

6.2.2 *run_metrics(data)*

This function is a helper function to automatically calculate all linkability metrics on all columns in *data*. The output will be collected and formatted into a Pandas DataFrame.

Parameters:

- *data* - The Pandas DataFrame to be modified. Please ensure that your data is in the correct format (as specified in Section 5.2) before passing as an argument.

Return Value:

- *metrics_df* - A Pandas DataFrame containing the values of every linkability metric calculated on every column in *data*. The columns of the DataFrame corrsepond to the linkability metrics, and the rows correspond to the names of the columns in *data.columns*.