Matching and Mapping for the TSDB (KairosDB)

SEED project

December 1, 2015

Contents

1 Introduction

The document records the process of implementing the mapping and matching operations for TSDB (KairosDB.

2 SEED Mapping and Matching

2.1 Mapping

Mapping rename the column/field names of the imported data set to terms in Building Energy Data Exchange Specification (BEDES) [6]. In the process, the program search through the terms in BEDES and returns a suggested field name for each of the imported field in the dataset. user can 1) choose which field they want to retain or ignore 2) modify the suggested mapping and input the BEDES term. During the input process, there is a list of 20 strings in the drop-down menu under the input bar each string contains the current input as a sub string (Figure 1).

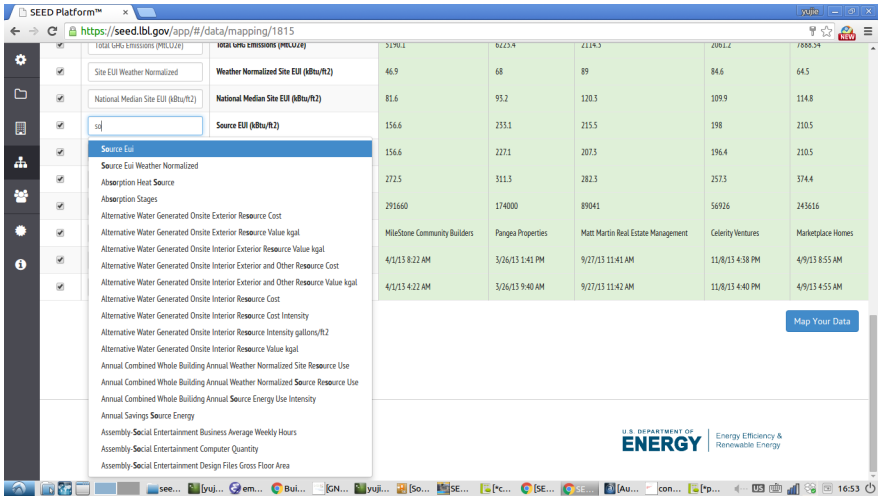

Figure 1: Drop-down list for input BEDES term

[4]

2.2 Matching

In the matching process, the fields in the two input tables, the building list and the PM data, are combined. The combining process utilizes some common fields from the two tables.

2.2.1 Automatic

The process uses fuzzy string searching [3,7] to auto-match the records in the two tables and returns a confidence score for the matching. In Figure 2 we can see the leading zeros does not affect the matching result.

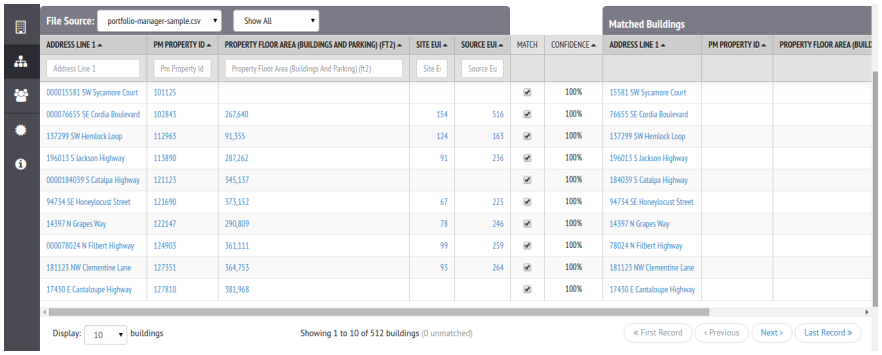

Figure 2: Matching result with confidence score
[4]

The guideline for the process is "to improve results in matching buildings across different data files, map as many of the following four (4) fields as possible: Tax Lot ID, PM Property ID, Custom ID, Address Line 1" [4].

2.2.2 Manual

The matching can be manually corrected by clicking on the value of the shared field in the source table and one can choose **one or more** records from the drop-down list that matches the record in the source table (Figure 3).

Figure 3: Manually correct matching result by selecting one or more potential record
[4]

In the matching process, one table is the source and the other is the target. For each record/row in the target table, if there exists a unique record in the source table that matches

this record, a match will be successful, but the score of confidence will not be 100% if there are multiple records in the target that matches the source.

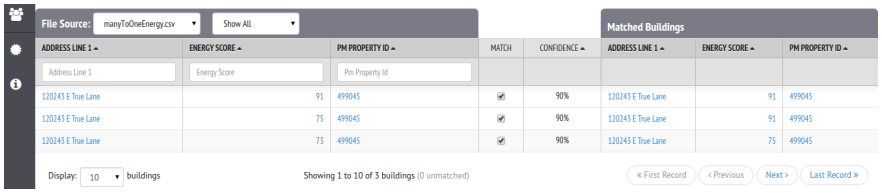

Figure 4: Three records in the target table (PM table) matches one record in the source table (Building table)

[4]

3 Implementation strategy of mapping and matching: approximate string matching

The approximate string matching aims at finding strings that *approximately* match some pattern. The matching is normally evaluated by some edit distance, which is the minimum number of primitive operations (e.g. insertion, deletion and substitution) needed to convert the approximate match to an exact match [7]. There are several versions of the set of primitive operations. One common definition is the Levenshtein distance, which include single-character operations as insertion, deletion and substitution.

There is a package in Python called FuzzyWuzzy [5] that evaluates the difference between strings with Levenshtein distance. The package is built upon the Python package difflib (which has a class called "SequenceMatcher" that compares two sequences (str, unicode, list, tuple, bytearray, buffer, xrange) as long as they are hashable (those that can become a dictionary key). Immutable types (number, string, tuples) are all hashable in Python). There are some explanation of the FuzzyWuzzy package here. The key functions include [2]:

```
from fuzzywuzzy import fuzz
# simple ratio: pure edit distance
# similar to difflib.SequenceMatcher
>>> fuzz.ratio("this is a test", "this is a test!")
    96

# partial ratio: when s1 and s2 have very different lengths
# WOLG, s1 < s2,
# partial_ratio(s1, s2) returns fuzz.ratio(s1, s2') where s2' is a
# sub string of s2 and len(s1) == len(s2')
```

```
>>> fuzz.partial_ratio("this is a test", "this is a test!")
    100

# token sort ratio:
# to deal with the word re-order of strings
break strings to tokens, sort tokens and then re-assemble them to strings before calculating t
fuzz.token_sort_ratio("fuzzy wuzzy was a bear", "wuzzy fuzzy was a bear")

# token set ratio:
# another way to deal with the word re-order of strings
# do not use it if duplicate words are important patterns
# let the input strings be s0 and s1
# t0 = intersection(sorted(s0), sorted(s1))
# s0' = sorted(s0) \ t0
# s1' = sorted(s1) \ t1
# t1 = intersection(t0, s0')
# t2 = intersection(t0, s1')
# return max(fuzz.ratio(t0, t1), fuzz.ratio(t1, t2), fuzz.ratio(t0, t2))
>>> fuzz.token_sort_ratio("fuzzy was a bear", "fuzzy fuzzy was a bear")
    84
>>> fuzz.token_set_ratio("fuzzy was a bear", "fuzzy fuzzy was a bear")
    100

# extracting a list of tuples of (str, score) where str is in choices
# and score is the matching score between query and choice)
# scorer: the ratio calculation method, can also be user defined
# limit: length of the returned list
>>> extract(query, choices, processor=None, scorer=None, limit=5)

>>> choices = ["Atlanta Falcons", "New York Jets", "New York Giants", "Dallas Cowboys"]
>>> process.extract("new york jets", choices, limit=2)
    [('New York Jets', 100), ('New York Giants', 78)]
```

However, if the address line is selected as the field for matching calculation, a **substitution of common abbreviations** should be performed before the string searching process

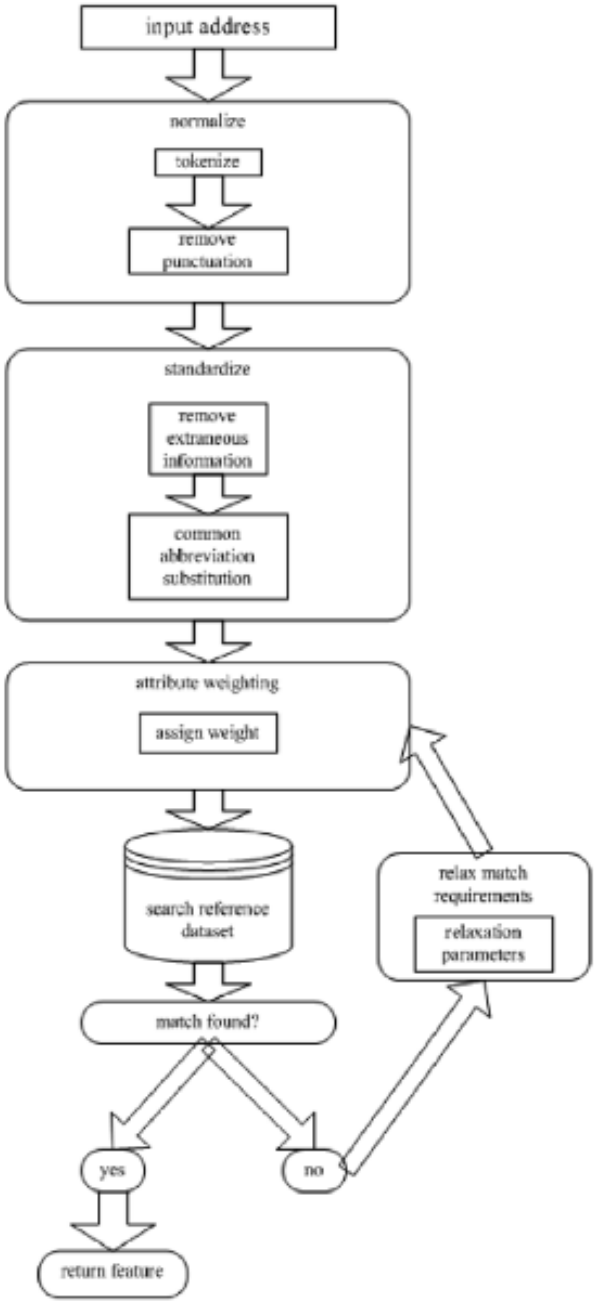

Figure 5: Schematic of deterministic address matching [1]

4 Progress

Assuming the matching field is "Address Line 1"

- Implemented a pre-process routine for input address strings with
  - Removing trailing dots

– Common abbreviation upper-case conversion and look-up (source: `http://pe.usps.gov/text/pub28/28apc_002.htm`)

```
WLS -> WLS
CPE -> CPE
ORCHRD -> ORCH
CRESCENT -> CRES
FALL -> FALL
BEACH -> BCH
MSSN -> MSN
RAMP -> RAMP
KYS -> KYS
SPG -> SPG
JCTN -> JCT
TUNEL -> TUNL
PARKWAYS -> PKWY
COVE -> CV
BYP -> BYP
SPRINGS -> SPGS
ISLANDS -> ISS
RIVER -> RIV
SPUR -> SPUR
JCTS -> JCTS
VIADCT -> VIA
PINES -> PNES
EXPRESS -> EXPY
MNRS -> MNRS
TUNLS -> TUNL
GROVES -> GRVS
SUMITT -> SMT
OVL -> OVAL
VIEW -> VW
CRSNT -> CRES
...
```

Output of preProcess

	input	output
0	5000 Forbes Ave.	5000 Forbes AVE
1	399 Asterisk Way	399 Asterisk WAY
2	Miller African-Centered Academy	Miller African-Centered Academy
3	5468 Bartlett St	5468 Bartlett ST
4	PNC Park, 115 Federal St	PNC Park, 115 Federal ST
5	606A P J McArdle Roadway	606A P J McArdle Roadway
6	411 7th Ave #360	411 7th AVE #360
7	436 Seventh Ave	436 Seventh AVE
8	3915 O'Hara St	3915 O'Hara ST
9	3302 WWPH	3302 WWPH

```
10                 5200 Pembroke Pl                 5200 Pembroke PL
11                      2 Bayard Rd                      2 Bayard RD
```

- Wrote a wrapper of matching function with FuzzyWuzzy package

```
test_similarity(5000 Forbes Avenue, 5000 Forbes Ave)
original: 5000 Forbes Avenue, after preProcess: 5000 Forbes AVE
match ratio with simple is 87
match ratio with partial is 87
match ratio with token_sort is 100
match ratio with token_set is 100
```

5 Next Step

Next step is to know which field to match (address or not) find a set of testing source and target strings to see if the matching function works well.

References

[1] D. W. Goldberg, J. P. Wilson, and Knoblock. From Text to Geographic Coordinates: The Current State of Geocoding. *Journal of the Urban and Regional Information Systems Association*, 19(1):33–46, 2007.

[2] JeffPaine et al. seatgeek/fuzzywuzzy. web, November 2015. `https://github.com/seatgeek/fuzzywuzzy`.

[3] Lawrence Berkeley National Laboratory. Seed 1.1 tutorial. web, November 2015. `https://windows.lbl.gov/projects/SEED/IntroTutorial/SEED%201.2%20Overview.htm`.

[4] Lawrence Berkeley National Laboratory. Seed platform. web, November 2015. `https://seed.lbl.gov/app/#/data/mapping/1815`.

[5] Python Software Foundation. fuzzywuzzy 0.8.0. web, November 2015. `https://pypi.python.org/pypi/fuzzywuzzy`.

[6] U.S. Department of Energy. Bedes. web, November 2015. `https://bedes.lbl.gov/`.

[7] Wikipedia. Approximate string matching. web, November 2015. `https://en.wikipedia.org/wiki/Approximate_string_matching`.