Address Fuzzy Matching

Project 2 Milestone Report Report2

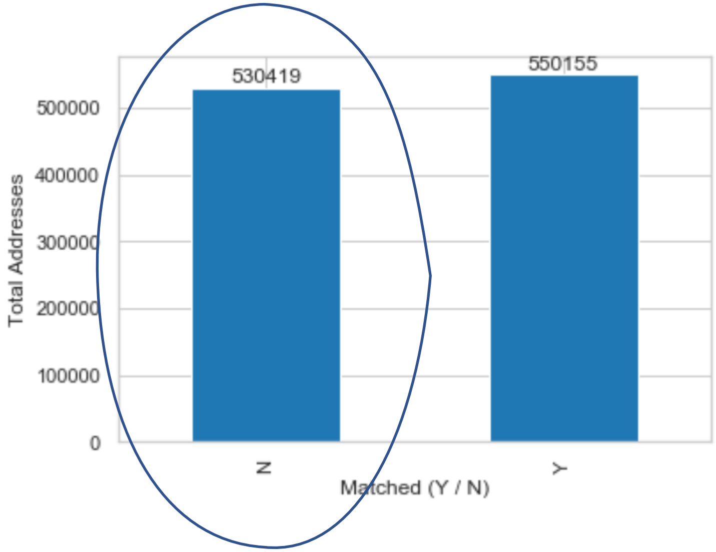
Overview

- Problem
- The data
- Data Wrangling and preparation
- Exploratory Data Analysis
- Machine Learning

Problem

- Background: All service installation records are stored and maintained in a file store with address info. However, no system or field linkage is built between the file store and ERP application. The address is entered manually each time the file added to the file store. The only way to find the records for an address in ERP is to manually search the file store using the address.

- Problem: Fuzzy match two sets of addresses



The Data

Two sets of address data

- Set 1 is the list of addresses from a legacy system

```
Data columns (total 6 columns):
DATAID         530437 non-null object
M_TXT          530437 non-null object
M_DIV_CD       530437 non-null object
M_ADDR         526190 non-null object
M_CITY         530320 non-null object
M_ADDR_TYPE    530401 non-null object
dtypes: object(6)
```

- Set 2 is the list of addresses from address master table

```
Data columns (total 3 columns):
P_DIV_CD     505767 non-null object
P_ADDR       505761 non-null object
P_CITY       505767 non-null object
dtypes: object(3)
```

- USPS Suffix data

Primary_Suffix	Comm_suffix	Std_suffix
alley	allee	aly
alley	alley	aly
alley	ally	aly
alley	aly	aly

```
Data columns (total 3 columns):
Primary_Suffix    515 non-null object
Comm_suffix       515 non-null object
Std_suffix        515 non-null object
dtypes: object(3)
```

Data Wrangling

Perform text normalization includes:

- converting all letters to lower case
- Removing punctuations
- Removing non ascii chars
- Removing multiple spaces with a single space

```python
def cleanse_str(string):
    string = ftfy.fix_text(string) # fix text encoding issues
    string = string.encode("ascii", errors="ignore").decode() #remove non ascii chars
    string = string.lower() # make lower case
    chars_to_remove = ["#", "@", ")","(",".","|","[","]","{","}","'"]
    rx = '[' + re.escape(''.join(chars_to_remove)) + ']'
    string = re.sub(rx, '', string) # remove the list of chars defined above
    string = string.replace('&', 'and')
    string = string.replace(',', ' ')
    string = string.replace('-', ' ')
    string = string.replace('+', ' and ')
    string = re.sub(' +',' ',string).strip() # get rid of multiple spaces and replace with a single space
    string = ' '+ string +' ' # pad names for ngrams...
    string = re.sub(r'[,-./]|\sBD',r'', string)

    return string
```

Drop rows if addresses are blank or numeric

```
Data columns (total 12 columns):
DATAID            525887 non-null object
M_TXT             525887 non-null object
M_DIV_CD          525887 non-null object
M_ADDR            525887 non-null object
M_CITY            525887 non-null object
M_ADDR_TYPE       525887 non-null object
M_ADDR_c          525887 non-null object
M_TXT_c           525887 non-null object
M_ADDR_n          525887 non-null object
ADDR_isnumeric    525887 non-null bool
M_ADDR_s          525887 non-null object
M_TXT_s           525887 non-null object
dtypes: bool(1), object(11)
```

Feature engineering

- Computing and adding columns for data exploration and machine learning
- Prepare data sets including legacy data, master data and USPS suffix data

```
Data columns (total 16 columns):
DATAID         30000 non-null int64
M_TXT          30000 non-null object
M_DIV_CD       30000 non-null object
M_ADDR         30000 non-null object
M_CITY         30000 non-null object
M_ADDR_TYPE    30000 non-null object
P_ID           30000 non-null int64
P_ADDR         30000 non-null object
P_CITY         30000 non-null object
MATCHCODE      30000 non-null object
M_ADDR_c       30000 non-null object
P_ADDR_c       30000 non-null object
M_TXT_c        30000 non-null object
M_ADDR_s       30000 non-null object
P_ADDR_s       30000 non-null object
M_TXT_s        30000 non-null object
dtypes: int64(2), object(14)
```

Data Wrangling – Suffix consolidation

Analyze suffix variances and usages

- Split address into words
- Use Collection Counter to calculate the word occurrences
- Use USPS suffix data to identify suffix words
- Generate suffix variance matrix

```python
# generate matrix for top 10 primary suffix which have multiple variances in legacy address data set
include = list(result1.groupby("Primary_Suffix").Word_Seg.count().sort_values(ascending=False).head(10).index)
df_m_suffix = result1.query('Primary_Suffix in @include').sort_values(['Primary_Suffix'], ascending=True)
df_m_suffix.groupby(['Primary_Suffix','Word_Seg']).size().unstack(fill_value=0)
```

Word_Seg Primary_Suffix	av	ave	aven	avenue	cent	center	centr	centre	cir	circl	...	ridge	st	sta	station	stn	str	street	ter	terr	terrace
avenue	1	1	1	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
center	0	0	0	0	1	1	1	1	0	0	...	0	0	0	0	0	0	0	0	0	0
circle	0	0	0	0	0	0	0	0	1	1	...	0	0	0	0	0	0	0	0	0	0
drive	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
heights	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
highway	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
ridge	0	0	0	0	0	0	0	0	0	0	...	1	0	0	0	0	0	0	0	0	0
station	0	0	0	0	0	0	0	0	0	0	...	0	0	1	1	1	0	0	0	0	0
street	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	0	1	1	0	0	0
terrace	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	1	1	1

Prepare data sets

- Standardize suffix variances into USPS primary suffix

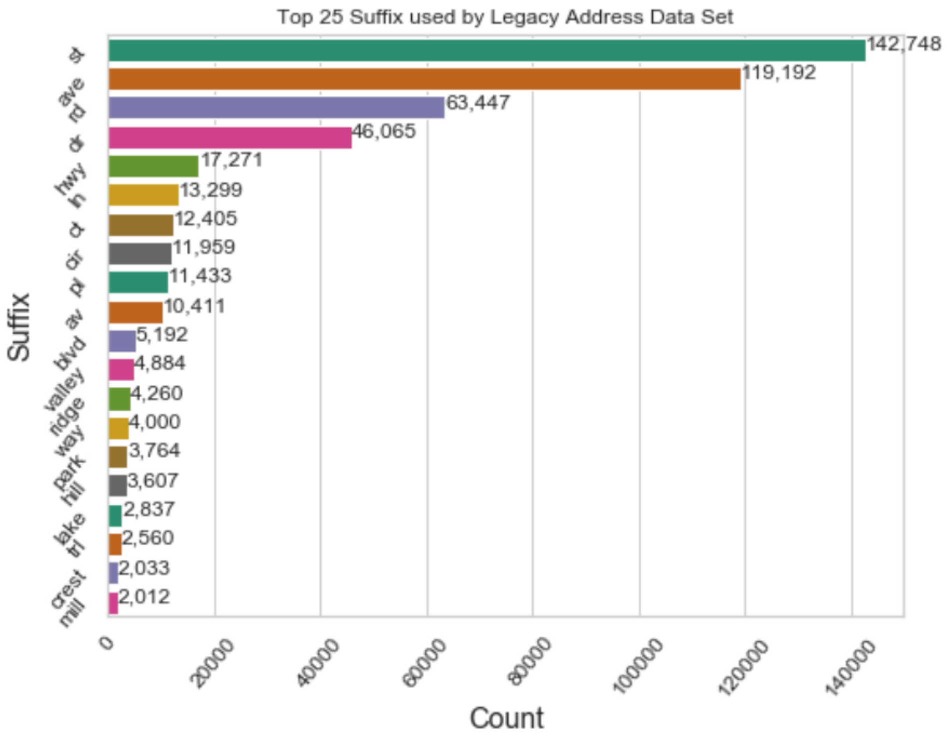
```python
string = re.sub(' aly ', ' alley ', string)
string = re.sub(' annex ', ' anex ', string)
string = re.sub(' anx ', ' anex ', string)
string = re.sub(' ave ', ' avenue ', string)
string = re.sub(' av ', ' avenue ', string)
string = re.sub(' aven ', ' avenue ', string)
string = re.sub(' bch ', ' beach ', string)
string = re.sub(' bnd ', ' bend ', string)
string = re.sub(' blf ', ' bluff ', string)
string = re.sub(' btm ', ' bottom ', string)
string = re.sub(' blvd ', ' boulevard ', string)
```

Word_Seg	cnt
center	933
ctr	338
centre	14
cent	94
cntr	7
centr	1

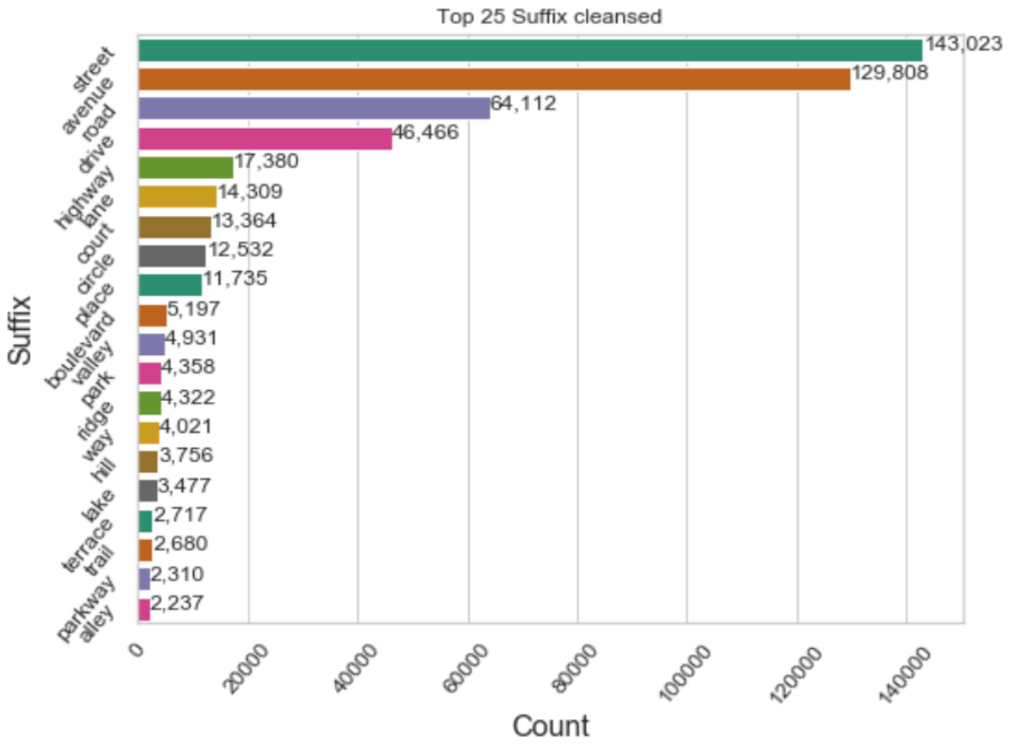
Word_Seg	cnt
center	1387

Exploratory Data Analysis – Suffix consolidation

Before suffix consolidation



After suffix consolidation



Exploratory Data Analysis – verify improvement of suffix

Run Fuzzywuzzy functions against two data sets and compare the matching results

- Ratio (result: r1, r1s)
- partial_ratio (pr1, pr1s)
- token_sort_ratio (tsr1, tsr1s)
- token_set_ratio (tstr1, tstr1s)

Among four Fuzzywuzzy functions, token_set_ratio returns the best matching ratio 93% (ratio > 90)

The matching score increased 2.25% for the data with consolidated suffix

Run 1

Data set – cleansed with raw suffix

```
r1 match (%>90) =  0.8581
pr1 match (%>90) =  0.8755
tsr1 match (%>90) =  0.8383666666666667
tstr1 match (%>90) =  0.908
```

Run 2

Data set – cleansed with suffix consolidated

```
r1s match (%>90) =  0.8698333333333333
pr1s match (%>90) =  0.8921
tsr1s match (%>90) =  0.8601
tstr1s match (%>90) =  0.9305333333333333
```

Exploratory Data Analysis– verify improvement of suffix

Data Set: samples of matched addresses

DLP – Fuzzywuzzy token_set_ratio

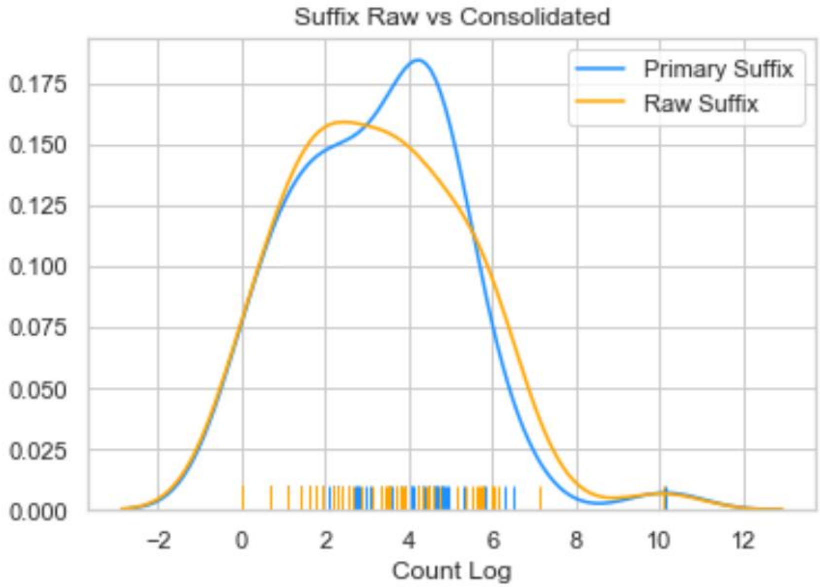
Fuzzy matching using raw suffix

- Features
  - M_ADDR_c
  - P_ADDR_c

Fuzzy matching using consolidated suffix

- Features
  - M_ADDR_s
  - P_ADDR_s

```
Data columns (total 16 columns):
DATAID          30000 non-null int64
M_TXT           30000 non-null object
M_DIV_CD        30000 non-null object
M_ADDR          30000 non-null object
M_CITY          30000 non-null object
M_ADDR_TYPE     30000 non-null object
P_ID            30000 non-null int64
P_ADDR          30000 non-null object
P_CITY          30000 non-null object
MATCHCODE       30000 non-null object
M_ADDR_c        30000 non-null object
P_ADDR_c        30000 non-null object
M_TXT_c         30000 non-null object
M_ADDR_s        30000 non-null object
P_ADDR_s        30000 non-null object
M_TXT_s         30000 non-null object
dtypes: int64(2), object(14)
```



Machine Learning

DLP Algorithms

- Fuzzywuzzy (Levenshtein Distance)
- KNN - K Nearest Neighbor (tf-idf / Distance)
- TopN - by ING (tf-idf / Cosine Similarity)

Features

- M_DIV_CD
- M_ADDR_c
- M_ADDR_s
- P_DIV_CD
- P_ADDR_c
- P_ADDR_s

Metrics measure the match results

- KNN – Match Confidence Score
- TopN - Similarity score
- Fuzzywuzzy - Ratio

Identify algorithm works best for the data

Data Set

- ds1 - Samples of matched addresses
- ds2 – Master addresses

DLP Algorithm

- KNN
- TopN
- Fuzzywuzzy token_set_ratio

Features

- M_DIV_CD
- P_DIV_CD
- M_ADDR_s
- P_ADDR_s

```
ds1 = df_match[df_match.M_DIV_CD=='d6'].M_ADDR_s.unique()
ds2 = df_master[df_master.P_DIV_CD=='d6'].P_ADDR_s.unique()
```

```
len(ds1), len(ds2)
```

```
(1107, 19023)
```

Fuzzywuzzy

```
Total time consumed: 152.90584063529968
tstr1s match (%>90) =  0.9611562782294489
```

KNN

```
Total time consumed: 4.798900365829468
r1s match (%>90) =  0.7199638663053297
pr1s match (%>90) =  0.8518518518518519
tsr1s match (%>90) =  0.8970189701897019
tstr1s match (%>90) =  0.9376693766937669
```

TopN

```
Total time consumed: 0.7635231018066406
r1s match (%>90) =  0.8925022583559169
pr1s match (%>90) =  0.8762420957542909
tsr1s match (%>90) =  0.8970189701897019
tstr1s match (%>90) =  0.9376693766937669
```

Fuzzywuzzy returns 96% matches with ratio>90% which is higher than KNN and topN; however, the computing cost is much higher than KNN and topN

KNN and topN return 94% matches with ratio>90% with much lower computing cost

Identify algorithm works best for the data – cont.

This data set is matched addresses set.

Observations:

Fuzzywuzzy ratio best describe the matching results.

Token_set_ratio returns the best matching ratio 93% (ratio > 90)

KNN and topN returns the nearest neighbors and top 1 match with the highest confidence and closest similarity. However, the Similarity score and confidence score are not in line with the matching quality.

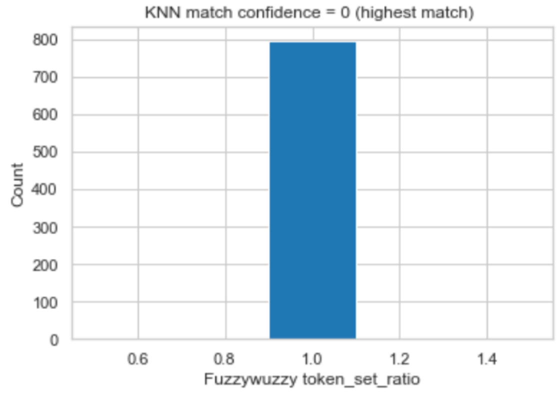
Conclusion:

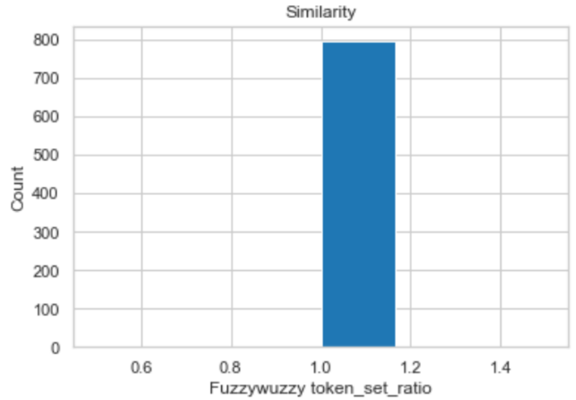
KNN and topN can be used as means to find the closest / top matches

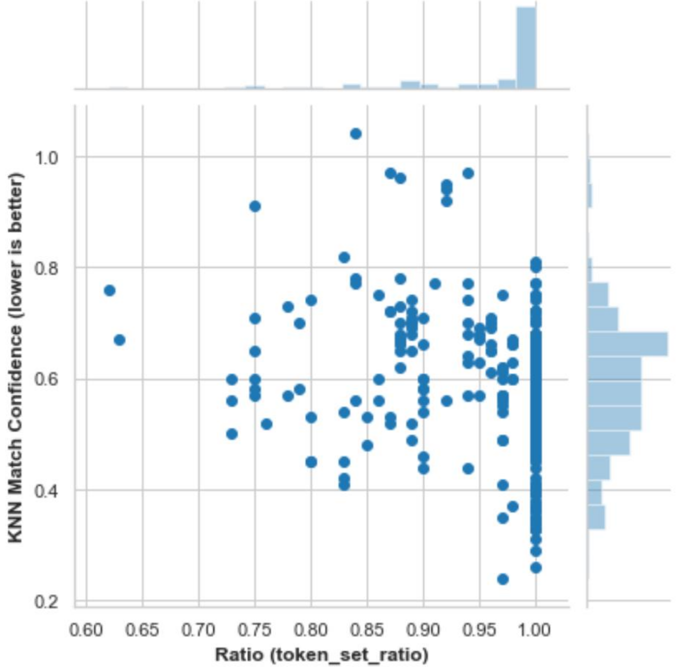
Fuzzywuzzy can be used to validate the matching results and set the threshold of good matches

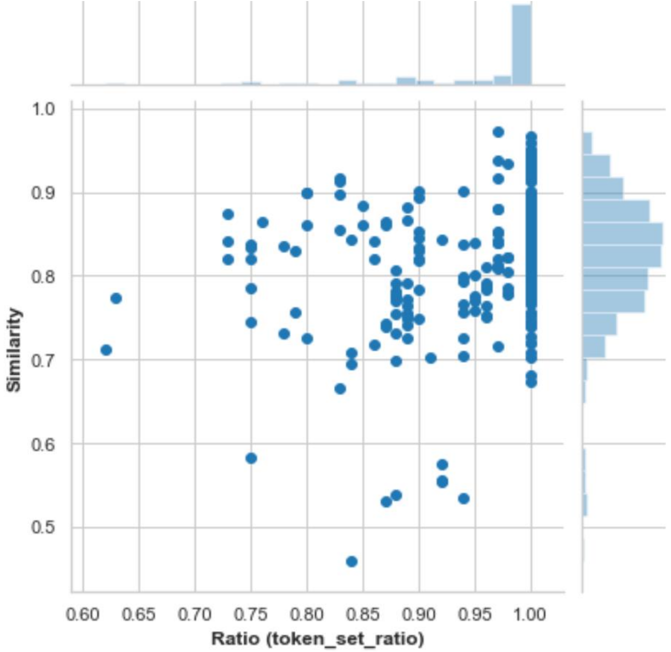
Token_set_ratio best suits the data

Compare Confidence / Similarity and Ratio









TopN scalability test results

Data Set

- ds1 – Messy addresses
- ds2 – Master addresses

DLP Algorithm

- TopN
- Fuzzywuzzy

Features

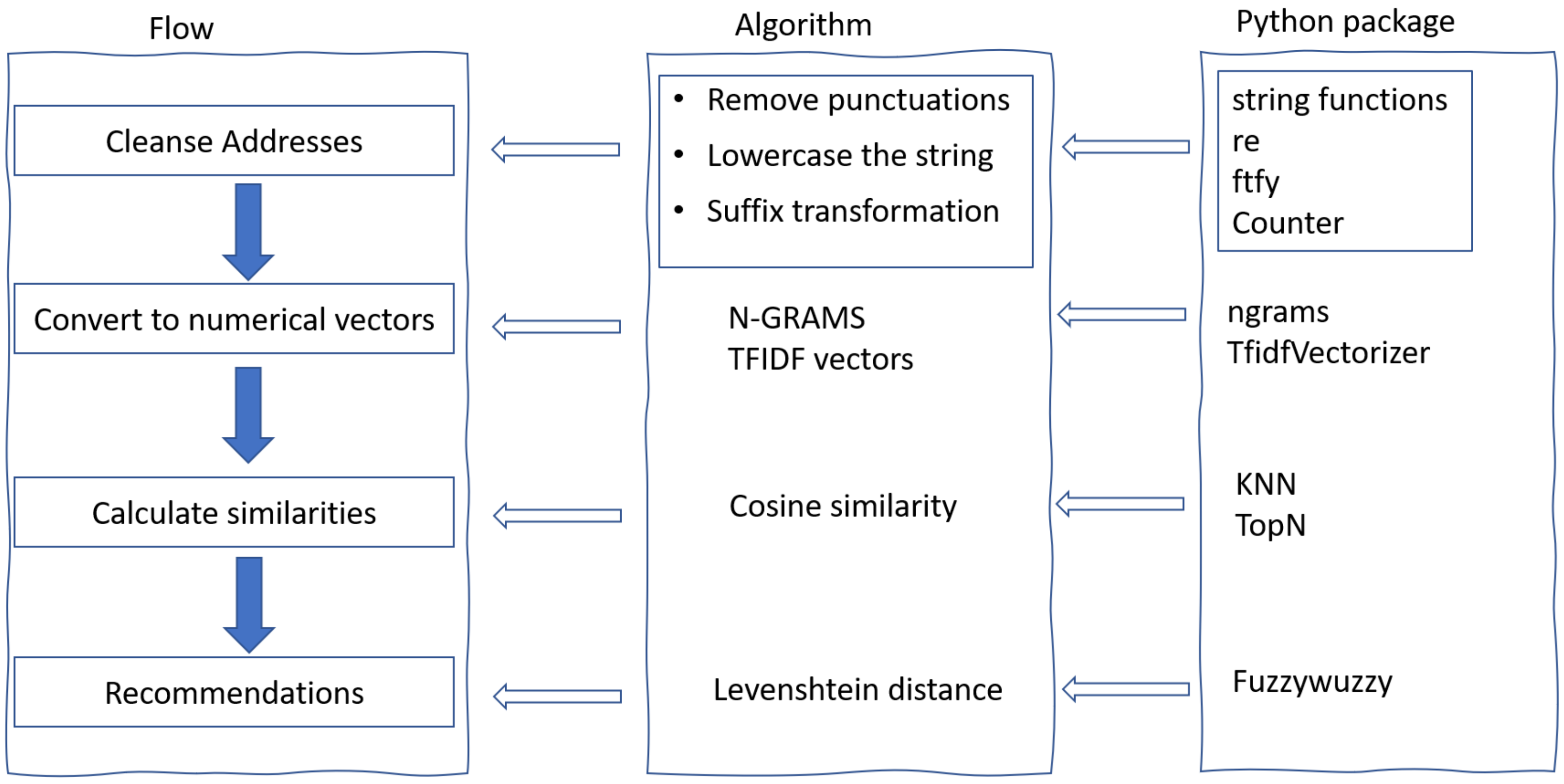
- M_DIV_CD
- P_DIV_CD
- M_ADDR_s
- P_ADDR_s

Data Size	Results	Total time
2000	r1s match (%>90) = 0.492 pr1s match (%>90) = 0.4875 tsr1s match (%>90) = 0.4555 tstr1s match (%>90) = 0.592 ING program completed. Time used: 1.2478554248809814	1.24 seconds
11864	r1s match (%>90) = 0.5670888913798902 pr1s match (%>90) = 0.5348055150767851 tsr1s match (%>90) = 0.5246048649254568 tstr1s match (%>90) = 0.6196614729290438 ING program completed. Time used: 3.764163017272949	3.76 seconds
317149	Total time consumed: 3773.7916209697723 r1s match (%>90) = 0.5065046372501164 pr1s match (%>90) = 0.526466948950517 tsr1s match (%>90) = 0.42622685631903373 tstr1s match (%>90) = 0.5728933716270674	1 hour

Architecture

Based on the above findings, a fuzzy matching solution is designed as below

- Use topN algorithm to perform fast fuzzy match
- Use Fuzzywuzzy to validate the matched pairs
- Use Fuzzywuzzy ratio as indicator for recommendations

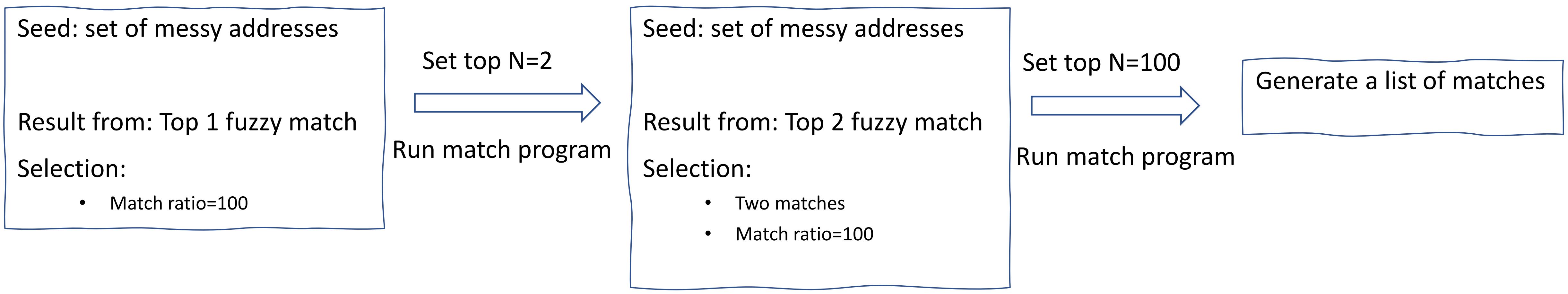


Architecture for multiple matches

One address could match to multiple address. One example is that one street address matches to a set of apartment complex addresses.

N is variable which can be set to any number applicable. If an address found two matches with ratio = 100, then there is potential that more matches could be found. A threshold 100 is set for this data set.

Set N to 2 and 100 for two runs will greatly improve the performance and still generate the desired results



Final run

Data Set

- ds_1 – All unmatched addresses
- ds_2 – Master addresses

DLP Algorithm

- TopN
- Fuzzywuzzy token_set_ratio

Features

- M_DIV_CD
- P_DIV_CD
- M_ADDR_s
- P_ADDR_s

```
M_DIV_CD
d1     57665
d2    280559
d3     34734
d4     54960
d5     22550
d6     21533
d7     53886
Name: DATAID, dtype: int64
```

```
Data columns (total 8 columns):
Messy_addr     317149 non-null object
Master_addr    317149 non-null object
Similarity     317149 non-null float64
pr1s           317149 non-null int64
tsr1s          317149 non-null int64
tstr1s         317149 non-null int64
r1s            317149 non-null int64
M_DIV_CD       317149 non-null object
dtypes: float64(1), int64(4), object(3)
```

```
ds_master = df_master[df_master.P_DIV_CD==div_cd][master_feature].unique()
ds_messy = df_exception[df_exception.M_DIV_CD==div_cd][messy_feature].unique()
```

```
for div_cd in div_list:

    m_t1, m_tn, m_tn_seed = get_div_match (div_cd, master_feature, messy_feature)
    m_t1s = m_t1s.append(m_t1)
    m_tns = m_tns.append(m_tn)
    m_tn_seeds = m_tn_seeds.append(m_tn_seed)
```

Run fuzzy match by division. Two benefits:

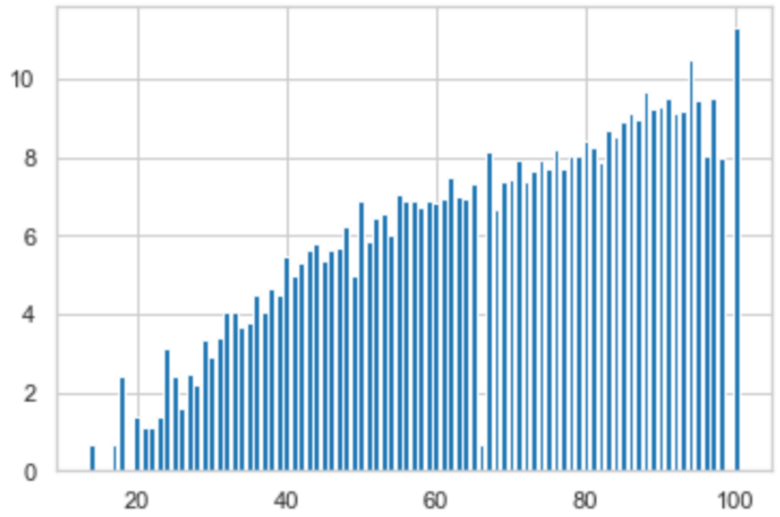
- Keep DIV_CD as a feature
- Break large data set into smaller size

The total run time improved from 1 hour to 20 minutes.

```
The run completed in: 1210.2415356636047
size of m_t1s: 317149
size of m_tns: 86395
size of m_tn_seeds: 5959
```

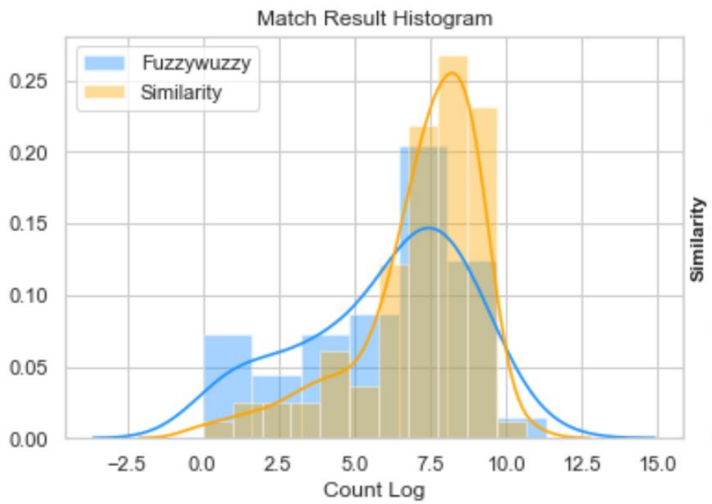
Final run results

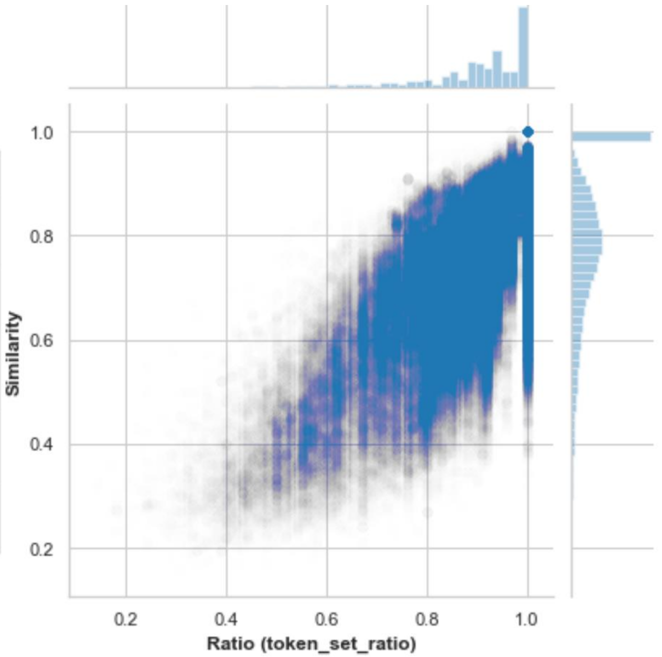
Fuzzywuzzy token_set_ratio ratio bar plot




```
len(matches_all[matches_all.tstr1s==100])/len(matches_all)
```

```
0.255961708849783533
```






```
df_export_top1.to_csv('DLP_Matched_Addr_top1.csv')
m_tns.to_csv('DLP_Matched_Addr_multiple_raw.csv')
df_export_multiple.to_csv('DLP_Matched_Addr_multiple.csv')
df_master.to_csv('DLP_Master_cleansed.csv')
```

Conclusion

- An efficient fuzzy match program is built
- Initial fuzzy match results
  - Ratio=100 returns about 25.6% match with very good confidence (the data is very dirty which couldn’t use the third-party address tool to resolve)
  - Fuzzy match by similarity (with ratio<100) also set the initial data set for further analysis, validation and clean up.

- Next Steps:
  - There is room to improve match% by lower ratio from 100. Need more analysis on threshold of ratio and similarity
  - Cleanse and add additional features, i.e., city
  - Explore ngrams algorithm to see if any changes to generate grams can improve matching accuracy and score