CS 838 (Spring 2017) - Data Science
Project Stage - 4 Report

<u>Group : 17</u>
Gautam Singh
Harsh Singhal
Naman Agrawal

Objective : Combining data into a single set

In this project stage, we find the set of matching tuple pairs using supervised learning based entity matcher developed in previous project stage and then merge matched tuple pairs to form a single table.

1. **How did you combine the two tables A and B to obtain E? Did you add any other table? When you did the combination, did you run into any issues?**

Schema Matching:

In previous project stage, we created tables A(yelp.csv) and B(zomato.csv) using data obtained from Yelp and Zomato website. Both tables A and B have same schema and each tuple in both table describe entity type restaurant with attributes -

< ID, Name, Phone, Zipcode, State, City, Address, Delivery, Takeout, Outdoor_seating >

Since, we have same schema for table A and B, and we opted not to add any other table, schema matching was easy.

However, we modified schema of table A and B to learn and demonstrate process of schema matching using jaccard similarity measure to find matching attribute pairs. Modified schema of table A and B have different name for some attributes as depicted below :

Table A : *<id, Name, Phone Number, Zipcode, State, City, Address, Has Delivery, Has Takeout, Outdoor_seating>*

Table B : *< id, Restaurant Name, Contact Number, Zipcode, State, City, Address, Delivery, Takeout, Outdoor seating >*

We ran into few data merging issues while combining tables A and B to obtain table E. Some issues have been tabulated below along with description .

Issue	Issue description
Restaurant name not matched	Many tuples had different name in table A and B. It was often the case when restaurant name was abbreviated in one of the table.
Phone number not matched	Many tuples had different phone number in table A and B. Many tuples have entirely two different phone number or number of phone numbers in the two table were different.
Address not matched	Many tuples had some fields in address like street names abbreviated due to which data didn't match. In many cases, order of different fields in address were different.
Delivery, Takeout, Outdoor seating field not matched	In very few cases these fields didn't match surprisingly. When these fields did not match we set its value as 'Unknown' .

Schema matching script : **Jupyter notebook**

In [1]:

```
import py_entitymatching as em #Import megallan entity matching library
import distance
```

In [7]:

```
# We have modified the schema of yelp and zomato for schema matching task.
yelp = em.read_csv_metadata("yelp_2.csv",key="id")
zomato = em.read_csv_metadata("zomato_2.csv",key="id")
```

In [3]:

```
# Yelp table schema
for attr in yelp.keys():
    print attr
```

```
id
Name
Phone Number
Zipcode
State
City
Address
Has Delivery
Has Takeout
Outdoor_seating
```

In [4]:

```
# Zomato table schema
for attr in zomato.keys():
    print attr
```

```
id
Restaurant Name
Contact Number
Zipcode
State
City
Address
Delivery
Takeout
Outdoor seating
```

In [5]:

```
# Carry out Schema Matching based on Jaccard distance and generate matching attribu
pairs = []
min_distance = 100
for yelp_attr in yelp.keys():
    for zomato_attr in zomato.keys():
        dist = distance.jaccard(yelp_attr,zomato_attr)
        if dist < min_distance:
            min_distance = dist
            selected_attribute = zomato_attr
    pairs.append([yelp_attr,selected_attribute])
    min_distance = 100
```

In [6]:

```
# Display the matched attribute pairs
pairs
```

Out[6]:

```
[['id', 'id'],
 ['Name', 'Restaurant Name'],
 ['Phone Number', 'Contact Number'],
 ['Zipcode', 'Zipcode'],
 ['State', 'State'],
 ['City', 'City'],
 ['Address', 'Address'],
 ['Has Delivery', 'Delivery'],
 ['Has Takeout', 'Takeout'],
 ['Outdoor_seating', 'Outdoor seating']]
```

Data Merging:

We formulated data merging rules based on issue observed while combining table A and B. We devised strategy to merge data for each attribute .

- ***Name***

We selected restaurant name with more length because name with lesser length was often abbreviated name.

- ***Phone***

We compared phone numbers after removing special characters. If numbers matched we selected the number for combined table otherwise phone numbers were combined as comma separated list and added to phone number column.

- ***Zipcode, State, City***

Since blocking was done based on exact match for ZipCode, State and City; we selected these values from yelp table to merge in table E.

- ***Address***

We chose to select address having higher length in order to keep as much information as possible. We noticed that address with shorter length often had abbreviated street name or some fields like landmark were missing.

- ***Delivery, Takeout, Outdoor_seating***

We chose to set value for these fields to be ‘Unknown’ when it didn’t match between two matching tuples. We also replaced binary representation for these fields by Yes/No/ Unknown .

2. What is the schema of Table E, how many tuples are in Table E?

Table E contains 272 tuples and schema for table E is as follows :

< ID, Restaurant_name, Phone, Zipcode, State, City, Address, Delivery, Takeout, Outdoor_seating >

Few sample tuples from table E are listed below :

ID	Restaurant_name	Phone	Zipcode	State	City	Address	Delivery	Takeout	Outdoor_seating
512	McSorley's Old Ale House	(212) 473-9148	10003	NY	New York	15 E 7th Street	No	No	No
4	El Techo de Lolinda	(415) 550-6970	94110	CA	San Francisco	2518 Mission District Street	No	Unknown	Yes
1030	Phat Philly Cheesesteaks	(415) 550-7428	94110	CA	San Francisco	3388 24th Street		Yes	Yes
1031	Samovar Tea Lounge	(415) 227-9400	94103	CA	San Francisco	30 Howard Street		Yes	Yes

3. <u>Python script</u> that we used to merge the two tables A and B.

Python script : Jupyter notebook

In [1]:

```
import py_entitymatching as em #Import megallan entity matching library
```

In [2]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:

```
def phone_match(str1,str2):
    if type(str1) is float and type(str2) is float:
        if math.isnan(str1) and math.isnan(str2):
            return True
    elif type(str1) is float:
        if math.isnan(str1):
            return False
    elif type(str2) is float:
        if math.isnan(str2):
            return False
    else:
        stra = ""
        strb = ""
        for ch in str1:
            if ch.isdigit():
                stra += ch
        for ch in str2:
            if ch.isdigit():
                strb += ch
        if stra == strb:
            return True
        else:
            return False
```

In [4]:

```
# Import the data set after blocking
yelp = em.read_csv_metadata("yelp.csv",key="id")
zomato = em.read_csv_metadata("zomato.csv",key="id")
tagged_data = em.read_csv_metadata("tagged_dataset.csv", key='_id', fk_ltable='ltabl
```

```
No handlers could be found for logger "py_entitymatching.io.parsers"
```

In [5]:

```
# Selecting a subset(S) of tuples from the labelled_dataset - downsampling the tagge
# We decided not to down sample the data but have this code over here for use, if re
# Hence, we are copying the tagged_data to downsampled table (S)
S = em.sample_table(tagged_data, 900)
S = tagged_data
S.columns
```

Out[5]:

```
Index([u'key_id', u'_id', u'ltable_id', u'rtable_id', u'ltable_Name',
       u'ltable_Phone', u'ltable_Zipcode', u'ltable_State', u'ltable_C
ity',
       u'ltable_Address', u'ltable_Delivery', u'ltable_Takeout',
       u'ltable_Outdoor_seating', u'rtable_Name', u'rtable_Phone',
       u'rtable_Zipcode', u'rtable_State', u'rtable_City', u'rtable_Ad
dress',
       u'rtable_Delivery', u'rtable_Takeout', u'rtable_Outdoor_seating
',
       u'Label'],
      dtype='object')
```

In [6]:

```
S.shape
```

Out[6]:

```
(1100, 23)
```

In [7]:

```
# Split G into development (I) and evaluation (J)
IJ = em.split_train_test(S, train_proportion=.70)
I = IJ['train'] # Training Set
J = IJ['test'] # Test Set - not using
```

In [8]:

```
print "Number of tuples in Development Set =", len(I)
print "Number of tuples in Evaluation Set =", len(J)
```

```
Number of tuples in Development Set = 770
Number of tuples in Evaluation Set = 330
```

In [9]:

```
# Commenting this code section, since not required at this point of time

#Store Development Set
#I.to_csv('DevelopmentSet.csv')
#Store Evaluation Set
#J.to_csv('EvaluationSet.csv')
```

In [10]:

```
# Using Random Forrest for Machine Learning as it was found to be the best matcher .
rf = em.RFMatcher()
```

In [11]:

```
# Generate features
feature_set = em.get_features_for_matching(yelp, zomato)
feature_set.head(1)
```

Out[11]:

	feature_name	left_attribute	right_attribute	left_attr_tokenizer	right_attr_tokenizer	si
0	id_id_exm	id	id	None	None	ex

In [12]:

```
# Get feature vector table for Development set
I_feature_vectors = em.extract_feature_vecs(I, feature_table=feature_set, attrs_afte

# Get feature vector table for Evaluation set
J_feature_vectors = em.extract_feature_vecs(J, feature_table=feature_set, attrs_afte

I_feature_vectors.head(1)
```

Out[12]:

	_id	ltable_id	rtable_id	id_id_exm	id_id_anm	id_id_lev_dist	id_id_lev_sim	Name
319	10562	459	45	0	0.098039	1.0	0.666667	0.04

1 rows × 62 columns

In [13]:

```
# Fill the missing values with 0
I_feature_vectors.fillna(value=0, inplace=True)
tagged_data.columns
```

Out[13]:

```
Index([u'key_id', u'_id', u'ltable_id', u'rtable_id', u'ltable_Name',
       u'ltable_Phone', u'ltable_Zipcode', u'ltable_State', u'ltable_C
ity',
       u'ltable_Address', u'ltable_Delivery', u'ltable_Takeout',
       u'ltable_Outdoor_seating', u'rtable_Name', u'rtable_Phone',
       u'rtable_Zipcode', u'rtable_State', u'rtable_City', u'rtable_Ad
dress',
       u'rtable_Delivery', u'rtable_Takeout', u'rtable_Outdoor_seating
',
       u'Label'],
      dtype='object')
```

In [14]:

```
# Select the attrs. to be included in the feature vector table

attrs_from_table = ['ltable_Name', 'ltable_Phone', 'ltable_Zipcode', 'ltable_State',
                    'ltable_Delivery','ltable_Takeout','ltable_Outdoor_seating',
                    'rtable_Name', 'rtable_Phone', 'rtable_Zipcode', 'rtable_State',
                    'rtable_Delivery','rtable_Takeout','rtable_Outdoor_seating','Lab
                    ]
# Convert the cancidate set to feature vectors using the feature table
L = em.extract_feature_vecs(S, feature_table=feature_set, attrs_before=attrs_from_ta
```

In [15]:

```
# Get the attributes to be excluded while predicting
attrs_to_be_excluded = []
attrs_to_be_excluded.extend(['_id', 'ltable_id', 'rtable_id'])
attrs_to_be_excluded.extend(attrs_from_table)
```

In [16]:

```
rf.fit(table=I_feature_vectors, exclude_attrs=['_id','ltable_id', 'rtable_id', 'Labe

# Predict the matches
predictions = rf.predict(table=L, exclude_attrs=attrs_to_be_excluded,
              append=True, target_attr='predicted', inplace=False)

## Evaluate the result
eval_result_rf = em.eval_matches(predictions, 'Label', 'predicted')
em.print_eval_summary(eval_result_rf)
```

```
Precision : 99.64% (274/275)
Recall : 100.0% (274/274)
F1 : 99.82%
False positives : 1 (out of 275 positive predictions)
False negatives : 0 (out of 825 negative predictions)
```

In [17]:

```
predictions.head()
```

Out[17]:

	_id	ltable_id	rtable_id	ltable_Name	ltable_Phone	ltable_Zipcode	ltable_State	lt
0	13168	661	55	Oasis Cafe	(312) 443-9534	60602	IL	C
1	414520	2640	2063	La Taqueria	(415) 285-7117	94110	CA	S F
2	414500	2491	2063	El Farolito	(415) 824-7877	94110	CA	S F
3	414526	2659	2063	Taqueria Cancún	(415) 252-9560	94110	CA	S F
4	414536	2764	2063	El Techo	(415) 550-6970	94110	CA	S F

5 rows × 81 columns

In [18]:

```
predictions.shape
```

Out[18]:

```
(1100, 81)
```

In [19]:

```
# Get the attributes to be projected out
attrs_proj = []
#attrs_proj.extend(['_id', 'ltable_id', 'rtable_id'])
attrs_proj.extend(attrs_from_table)
attrs_proj.append('predicted')

# Project the attributes
predictions = predictions[attrs_proj]
```

In [20]:

```
predictions.head()
```

Out[20]:

	ltable_Name	ltable_Phone	ltable_Zipcode	ltable_State	ltable_City	ltable_Address	lt
0	Oasis Cafe	(312) 443-9534	60602	IL	Chicago	21 N Wabash Ave	0
1	La Taqueria	(415) 285-7117	94110	CA	San Francisco	2889 Mission St	0
2	El Farolito	(415) 824-7877	94110	CA	San Francisco	2779 Mission St	0
3	Taqueria Cancún	(415) 252-9560	94110	CA	San Francisco	2288 Mission St	0
4	El Techo	(415) 550-6970	94110	CA	San Francisco	2518 Mission St	0

In [21]:

```
predictions.shape
```

Out[21]:

```
(1100, 20)
```

In [22]:

```
predictions.to_csv("predictions.csv")
```

In [23]:

```
predictions.head(1)
```

Out[23]:

	Itable_Name	Itable_Phone	Itable_Zipcode	Itable_State	Itable_City	Itable_Address	It
0	Oasis Cafe	(312) 443-9534	60602	IL	Chicago	21 N Wabash Ave	0

In [24]:

```
# Add new columns in the table for the merged attributes

predictions['restaurant_name'] = None
predictions['phone'] = None
predictions['zipcode'] = None
predictions['state'] = None
predictions['city'] = None
predictions['address'] = None
predictions['delivery'] = None
predictions['takeout'] = None
predictions['outdoor_seating'] = None
```

In [ ]:

```
# Flushing the rows to CSV that contain matching tuples
indexes_to_keep = set()
index = 0

for index in range(predictions.shape[0]):
    tuple = predictions.iloc[index]
    if tuple['predicted'] == 1:
        indexes_to_keep.add(index)
    index += 1
sliced = predictions.take(list(indexes_to_keep))
sliced.to_csv("before_merging.csv") # Writing the resultant table to a CSV file.
```

Schema Merging

In [ ]:

```
indexes_to_keep = set()
index = 0

for index in range(predictions.shape[0]):
    tuple = predictions.iloc[index]
```

```
    if tuple['predicted'] == 1:

        # Merging the Names -
        # Picking the one that has more length
        if len(tuple['ltable_Name']) > len(tuple['rtable_Name']):
            tuple['restaurant_name'] = tuple['ltable_Name']
        else:
            tuple['restaurant_name'] = tuple['rtable_Name']

        # Merging the Phone no -
        phone1 = tuple['ltable_Phone']
        phone2 = tuple['rtable_Phone']
        if phone_match(phone1, phone2) is True: # When phone numbers are same
            tuple['phone'] = phone1
        else: # Case when phone nos are different. We keep both separated by comma.
            tuple['phone'] = phone1+ "," + phone2

        # Merging the Zipcode -
        # Since blocking was done based on exact match for ZipCode, picking the lef
        tuple['zipcode'] = tuple['ltable_Zipcode']

        # Merging the State -
        # Picking the left table attribute
        tuple['state'] = tuple['ltable_State']

        # Merging the City -
        # Picking the left table attribute
        tuple['city'] = tuple['ltable_City']

        # Merging the Address
        # Picking the one that has more length
        if len(tuple['ltable_Address']) > len(tuple['rtable_Address']):
            tuple['address'] = tuple['ltable_Address']
        else:
            tuple['address'] = tuple['rtable_Address']

        # Merging Delivery
        # If the value of ltable and rtable attributes differ, push "Unknown"
        # Else, use the left table attribute and push "Yes" for 1 and "No" for 0
        if tuple['ltable_Delivery'] != tuple['rtable_Delivery']:
            tuple['has_delivery'] = "unknown"
        else:
            if tuple['ltable_Delivery'] == 0:
                tuple['delivery'] = "No"
            else:
                tuple['delivery'] = "Yes"

        # Merging Takeout
        # If the value of ltable and rtable attributes differ, push "Unknown"
        # Else, use the left table attribute and push "Yes" for 1 and "No" for 0
        if tuple['ltable_Takeout'] != tuple['rtable_Takeout']:
            tuple['takeout'] = "unknown"
        else:
            if tuple['ltable_Takeout'] == 0:
```

```
            tuple['takeout'] = "No"
        else:
            tuple['takeout'] = "Yes"

    # Merging Outdoor seating
    # If the value of ltable and rtables attributes differ, push "unknown"
    # Else, use the left table attribute and push "Yes" for 1 and "No" for 0
    if tuple['ltable_Outdoor_seating'] != tuple['rtable_Outdoor_seating']:
        tuple['outdoor_seating'] = "unknown"
    else:
        if tuple['ltable_Outdoor_seating'] == 0:
            tuple['outdoor_seating'] = "No"
        else:
            tuple['outdoor_seating'] = "Yes"

    # Updating the tuple in predications table
    predictions.iloc[index] = tuple
    indexes_to_keep.add(index)

index += 1
```

In [ ]:

```
# Print the schema
predictions.head(1)
```

In [ ]:

```
# Fetch only those rows where predicted = "1" => get correctly matched tuples
sliced = predictions.take(list(indexes_to_keep))

# Drop columns before merging.
# Dropping old attributes

del sliced['ltable_Name']
del sliced['rtable_Name']
del sliced['ltable_Phone']
del sliced['rtable_Phone']
del sliced['ltable_Zipcode']
del sliced['rtable_Zipcode']
del sliced['ltable_State']
del sliced['rtable_State']
del sliced['ltable_City']
del sliced['rtable_City']
del sliced['ltable_Address']
del sliced['rtable_Address']
del sliced['ltable_Delivery']
del sliced['rtable_Delivery']
del sliced['ltable_Takeout']
del sliced['rtable_Takeout']
del sliced['ltable_Outdoor_seating']
del sliced['rtable_Outdoor_seating']
del sliced['Label'] # Dropping the column'Label'
del sliced['predicted'] # Dropping the column 'predicted

sliced.to_csv("filtered_predictions.csv") # Writing the resultant table to a CSV fi
```

In [ ]:

```
sliced.shape
```

In [ ]:

```
# Schema of the merged table
sliced.head(1)
```