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','Lak
                    ]
# 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	lta
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	Ita
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)
```