Combining Yelp and Zomato Data

Lokananda Dhage, Mary Feng, Varun Naik
CS 838 Project Stage 4

1. <u>Combining Yelp and Zomato Data</u>

a. *Re-doing Stage 3 for Yelp and Zomato Datasets (match_magellan.py)*
Since we performed entity matching on songs and tracks in the previous stage, for this stage, we performed entity matching on the restaurant datasets as well; essentially, we redid stage 3 for the restaurant datasets. Since our datasets were in json (yelp_restaurants.json and zomato_restaurants.json), we first converted from json to csv. This is done in jsontocsv.py, producing yelp_restaurants.csv (1403 tuples) and zomato_restaurants.csv (1759 tuples). Then, entity matching was done using Magellan in match_magellan.py, in a similar process as before. We added a column to yelp_restaurants.csv and zomato_restaurants.csv to generate A.csv and B.csv, respectively. We performed blocking to generate candidate C (3211 tuples). We sampled from C to create training set I (350 tuples) and testing set J (150 tuples). Based on the training and testing process, the best matcher was logistic regression. During training, using 10 fold cross validation, precision was 0.980000, recall was 0.990909, and F1 was 0.986561. Logistic regression was also the best matcher on the test set, with precision of 100%, recall of 97.62%, and F1 of 98.8%.

b. *Applying Logistic Regression Matcher to C (the set of candidate tuple pairs obtained after blocking on A and B) (merge_tables.py)*
Using the trained best matcher, logistic regression, matching was performed on C. The result is the set of all matches between A and B, Yelp and Zomato, respectively. Since the result is in the form of feature vectors, the result is merged with the original csv files to obtain the meaningful result. This is done in merge_tables.py, which produces matches.csv (938 tuples). We inspected matches.csv manually and removed false matches to create matches_cleaned.csv (911 tuples).

c. *Merging Yelp and Zomato to Obtain Table E (merge_tables.py)*
   i. Dropping Repetitive or Unnecessary Columns
   The result from the previous step, matches_cleaned.csv, contains 50 columns (16 columns from Yelp, 32 columns from Zomato, and 2 repetitive id columns for Yelp and Zomato). Some cleanup was performed. Repetitive columns were dropped: “ltable_business_id” is equivalent to “business_id” (from Yelp), “rtable_id” is equivalent to “id” (from Zomato). These repetitive columns were dropped and renamed to “yelp_id” and “zomato_id” for clarity. Additionally, the “locality_verbose” attribute was simply the value of the “locality” attribute + “Madison” so that

was repetitive. Finally, when doing the conversion from json to csv, the "location" attribute was extracted to separate columns, so that was also repetitive. Many columns were also dropped because they were unnecessary or did not offer helpful insight. Several columns had the same value for all tuples: "state" was all "WI", "type" was all "business", "R" was same as zomato_id, "switch_to_order_menu" was all 0, "currency" was all $, "offers" was all [], "has_online_delivery" was all 0, "is_delivering_now" was all 0, "has_table_booking" was all 0, "establishment_types" was all [], "city_y" was all Madison, "city_id" was all the same, and "country_id" was all the same. Other columns were not deemed helpful for analysis and thus dropped: "apikey", "url", "thumb", "photos_url", "menu_url", "featured_image", "deeplink", and "events_url".

ii. Zomato: Extracting Shorter Addresses from Full Addresses
Yelp addresses were not full addresses; they contained street number, but not city, state, or zipcode. Zomato addresses were full addresses, containing street number, city, state, and zipcode. This was repetitive. Therefore, the first part of the address was extracted from the full address. For instance, if the address was something like "123 Main Street, Madison, WI 53715", then the extracted shorter address would be "123 Main Street". This was some pre-processing done to help resolve between Yelp and Zomato addresses in a later step.

iii. Selecting Yelp over Zomato for City and Zipcode
The "city_x" attribute from Yelp was similar to the "locality" attribute from Zomato. Zomato's "locality" split the city of Madison into "Downtown Isthmus", "West Side", and "East Side", while Yelp simply had "Madison". We decided to take Yelp's "city_x" attribute (renamed to "city") over Zomato's "locality", so "locality" was dropped. Similarly, since Zomato had 9 tuples with missing zipcodes, we decided to drop the "zipcode" attribute from Zomato and take the "postal_code" attribute from Yelp, which was renamed "zipcode".

iv. The other possible candidate columns we considered for merging was stars(Yelp) and user_ratings(Zomato). However since we believe these these ratings give different information depending on the Yelp and Zomato users, we didn't merge these columns as there is no one correct and acceptable way of merging the user ratings from two sources.

The procedure to merge the Yelp and Zomato datasets are summarized below.

Column of table E	Original Columns (Yelp, Zomato)	Merging logic and rationale
name	name_x, name_y	The values in the resulting column (Name) was populated with the value from either the values in the columns name_x or name_y depending on whichever had more characters i.e greater the length of the string, we assumed it to be the more complete and accurate value. There were no case of missing values encountered.
address	address_x, address_y	Some pre-processing was done on the Zomato addresses to extract a shorter address from the original full addresses, as described previously. Similarly to name, the longer address was selected, as it would contain more information. For example, Yelp's address may be "123 Main Street", while Yelp's may be "123 Main Street Suite 100", which is more informative.
latitude	latitude_x latitude_y	As there was no way to determine the accuracy of the latitude values from either of the two columns. Hence, the values in the resulting column (Latitude) was populated with the average of the values of the two Latitude columns.
longitude	longitude_x longitude_y	As there was no way to determine the accuracy of the longitude values from either of the two columns. Hence, the values in the resulting column (Longitude) was populated with the average of the values of the two Longitude columns.
city	city_x, locality	The "city_x" attribute from Yelp was similar to the "locality" attribute from Zomato. Zomato's "locality" split the city of Madison into "Downtown Isthmus", "West Side", and "East Side", while Yelp simply had "Madison". We decided to take Yelp's "city_x" attribute (renamed to "city") over Zomato's "locality", so "locality" was dropped.
zipcode	postal_code, zipcode	Since the zipcode column from Zomato contained 9 tuples with missing values, we decided to drop Zomato's postal_code. We retained the postal_code column from Yelp and renamed it as zipcode.

2. **Statistics on Table E (E.csv)**

The schema for Table E contains 19 columns: yelp_id, zomato_id, neighborhood, city, zipcode, stars, review_count, is_open, attributes, categories, hours, cuisines, average_cost_for_two, price_range, user_rating, name, address, latitude, and longitude. There are 911 tuples in table E, which is E.csv. Here are five sample tuples from table E:

1) RJNAeNA-209sctUO0dmwuA,17502451,Capitol,Madison,53703.0,4.0,1236,1,"[u'Alcohol: full_bar', u""Ambience: {'romantic': False, 'intimate': False, 'classy': False, 'hipster': False, 'divey': False, 'touristy': False, 'trendy': False, 'upscale': False, 'casual': True}"", u'BikeParking: True', u'BusinessAcceptsCreditCards: True', u""BusinessParking: {'garage': False, 'street': True, 'validated': False, 'lot': False, 'valet': False}"", u'Caters: False', u'GoodForKids: True', u""GoodForMeal: {'dessert': False, 'latenight': False, 'lunch': True, 'dinner': True, 'breakfast': False, 'brunch': False}"", u'HasTV: True', u'NoiseLevel: loud', u'OutdoorSeating: True', u'RestaurantsAttire: casual', u'RestaurantsCounterService: True', u'RestaurantsDelivery: False', u'RestaurantsGoodForGroups: True', u'RestaurantsPriceRange2: 2', u'RestaurantsReservations: False', u'RestaurantsTableService: True', u'RestaurantsTakeOut: True', u'WheelchairAccessible: False', u'WiFi: free']","[u'American (Traditional)', u'Restaurants', u'German', u'American (New)', u'Steakhouses', u'Bars', u'Breakfast & Brunch', u'Salad', u'Nightlife']","[u'Monday 7:30-22:30', u'Tuesday 7:30-0:0', u'Wednesday 7:30-1:0', u'Thursday 7:30-1:0', u'Friday 7:30-1:30', u'Saturday 9:0-1:30', u'Sunday 9:0-22:30']","American, Breakfast, Burger",25,2,"OrderedDict([(u'aggregate_rating', u'4.8'), (u'rating_text', u'Excellent'), (u'rating_color', u'3F7E00'), (u'votes', u'1610')])",The Old Fashioned,23 N Pinckney St,43.0762316238,-89.3836456086
2) 6zZDTZ4ZZEYfN268iPz0uQ,17503679,Capitol,Madison,53703.0,4.5,38,1,"[u'Alcohol: full_bar', u'CoatCheck: True', u'NoiseLevel: loud', u'RestaurantsPriceRange2: 3', u'RestaurantsTableService: True', u'RestaurantsTakeOut: True', u'BusinessAcceptsCreditCards: True', u'GoodForKids: False', u'RestaurantsGoodForGroups: True', u'RestaurantsReservations: True', u""Ambience: {'romantic': False, 'intimate': False, 'classy': False, 'hipster': False, 'divey': False, 'touristy': False, 'trendy': False, 'upscale': False, 'casual': False}"", u'RestaurantsDelivery: False', u'BikeParking: True', u""BusinessParking: {'garage': False, 'street': False, 'validated': False, 'lot': False, 'valet': False}"", u'GoodForDancing: False', u""GoodForMeal: {'dessert': False, 'latenight': False, 'lunch': False, 'dinner': False, 'breakfast': False, 'brunch': False}"", u'HappyHour: True', u""Music: {'dj': False, 'background_music': True, 'no_music': False, 'karaoke': False, 'live': False, 'video': False, 'jukebox': False}"", u'OutdoorSeating: False', u'Smoking: no', u'HasTV: False', u'RestaurantsAttire: dressy']","[u'Bars', u'Sushi Bars', u'Restaurants', u'Nightlife', u'Lounges']","[u'Monday 11:30-14:30', u'Monday

16:30-22:0', u'Tuesday 11:30-14:30', u'Tuesday 16:30-22:0', u'Wednesday 11:30-14:30', u'Wednesday 16:30-22:0', u'Thursday 11:30-14:30', u'Thursday 16:30-22:0', u'Friday 16:30-22:0', u'Saturday 16:30-22:0', u'Sunday 16:30-22:0']","Asian, Japanese, Sushi",25,2,"OrderedDict([(u'aggregate_rating', u'4.2'), (u'rating_text', u'Very Good'), (u'rating_color', u'5BA829'), (u'votes', u'209')])",Red,"316 West Washington Ave, Ste 100",43.0738205,-89.384915

3) 0ETtpZSd6f7q_-o-7gutPg,17503369,Capitol,Madison,53703.0,4.5,264,1,"[u'Alco hol: beer_and_wine', u""Ambience: {'romantic': False, 'intimate': False, 'classy': False, 'hipster': True, 'divey': False, 'touristy': False, 'trendy': False, 'upscale': False, 'casual': False}"", u'BikeParking: True', u'BusinessAcceptsCreditCards: True', u""BusinessParking: {'garage': False, 'street': True, 'validated': False, 'lot': False, 'valet': False}"", u'ByAppointmentOnly: False', u'Caters: False', u'GoodForKids: True', u""GoodForMeal: {'dessert': False, 'latenight': False, 'lunch': False, 'dinner': False, 'breakfast': True, 'brunch': True}"", u'HasTV: False', u'NoiseLevel: average', u'OutdoorSeating: True', u'RestaurantsAttire: casual', u'RestaurantsDelivery: False', u'RestaurantsGoodForGroups: False', u'RestaurantsPriceRange2: 1', u'RestaurantsReservations: False', u'RestaurantsTableService: False', u'RestaurantsTakeOut: True', u'WheelchairAccessible: True', u'WiFi: free']","[u'Coffee & Tea', u'Restaurants', u'Creperies', u'Food']","[u'Monday 6:30-18:30', u'Tuesday 6:30-18:30', u'Wednesday 6:30-18:30', u'Thursday 6:30-18:30', u'Friday 6:30-18:30', u'Saturday 6:30-18:30', u'Sunday 6:30-18:30']","Breakfast, Coffee and Tea",10,1,"OrderedDict([(u'aggregate_rating', u'3.8'), (u'rating_text', u'Good'), (u'rating_color', u'9ACD32'), (u'votes', u'114')])",Bradbury's,127 N Hamilton St.,43.0769705,-89.3838785
4) kz5UANVmPxplIcoLnZBNhg,17503613,McClellan Park,Madison,53718.0,3.5,88,1,"[u'Alcohol: full_bar', u""Ambience: {'romantic': False, 'intimate': False, 'classy': False, 'hipster': False, 'divey': False, 'touristy': False, 'trendy': False, 'upscale': False, 'casual': True}"", u""BestNights: {'monday': False, 'tuesday': False, 'friday': True, 'wednesday': False, 'thursday': False, 'sunday': True, 'saturday': True}"", u'BikeParking: True', u'BusinessAcceptsBitcoin: False', u'BusinessAcceptsCreditCards: True', u""BusinessParking: {'garage': False, 'street': False, 'validated': False, 'lot': True, 'valet': False}"", u'Caters: True', u'CoatCheck: False', u'DogsAllowed: False', u'GoodForDancing: False', u'GoodForKids: True', u""GoodForMeal: {'dessert': False, 'latenight': False, 'lunch': True, 'dinner': True, 'breakfast': False, 'brunch': True}"", u'HappyHour: True', u'HasTV: True', u""Music: {'dj': False, 'background_music': False, 'no_music': False, 'karaoke': False, 'live': False, 'video': False, 'jukebox': False}"", u'NoiseLevel: average', u'OutdoorSeating: True', u'RestaurantsAttire: casual', u'RestaurantsDelivery: False', u'RestaurantsGoodForGroups: True', u'RestaurantsPriceRange2: 2', u'RestaurantsReservations: True', u'RestaurantsTableService: True', u'RestaurantsTakeOut: True', u'Smoking: no', u'WheelchairAccessible: True',

u'WiFi: free']","[u'Pubs', u'Nightlife', u'Breweries', u'Bars', u'Gift Shops', u'Gastropubs', u'Food', u'Shopping', u'Flowers & Gifts', u'Restaurants']","[u'Monday 11:0-2:0', u'Tuesday 11:0-2:0', u'Wednesday 11:0-2:0', u'Thursday 11:0-2:0', u'Friday 11:0-2:30', u'Saturday 11:0-2:30', u'Sunday 10:0-2:0']",American,25,2,"OrderedDict([(u'aggregate_rating', u'3.6'), (u'rating_text', u'Good'), (u'rating_color', u'9ACD32'), (u'votes', u'61')])",The Great Dane Pub & Brewing Co.,876 Jupiter Drive,43.08536,-89.28035

5) ubEa6XiMt6gOJ9xsUkbpEw,17503345,Capitol,Madison,53703.0,4.5,201,1,"[u'Alcohol: full_bar', u""Ambience: {'romantic': False, 'intimate': False, 'classy': True, 'hipster': False, 'divey': False, 'touristy': False, 'trendy': False, 'upscale': True, 'casual': False}"", u'BYOB: False', u'BYOBCorkage: no', u'BikeParking: True', u'BusinessAcceptsCreditCards: True', u""BusinessParking: {'garage': False, 'street': True, 'validated': False, 'lot': False, 'valet': False}"", u'Caters: False', u'DogsAllowed: False', u'GoodForKids: False', u""GoodForMeal: {'dessert': False, 'latenight': False, 'lunch': False, 'dinner': True, 'breakfast': False, 'brunch': False}"", u'HasTV: False', u'NoiseLevel: quiet', u'OutdoorSeating: False', u'RestaurantsAttire: dressy', u'RestaurantsCounterService: False', u'RestaurantsDelivery: False', u'RestaurantsGoodForGroups: True', u'RestaurantsPriceRange2: 4', u'RestaurantsReservations: True', u'RestaurantsTableService: True', u'RestaurantsTakeOut: False', u'WheelchairAccessible: True', u'WiFi: no']","[u'Restaurants', u'American (New)']","[u'Monday 17:30-0:0', u'Tuesday 17:30-0:0', u'Wednesday 17:30-0:0', u'Thursday 17:30-0:0', u'Friday 17:30-0:0', u'Saturday 17:0-0:0']","American, European, French",70,4,"OrderedDict([(u'aggregate_rating', u'4.2'), (u'rating_text', u'Very Good'), (u'rating_color', u'5BA829'), (u'votes', u'334')])",L'Etoile Restaurant,1 South Pinckney Street,43.0755377281,-89.3827955168

3. Python Code (merge_tables.py)

```
'''
Usage: python merge_tables.py
'''

import py_entitymatching as em
import match_magellan as mm
import pandas as pd

def get_all_matches():
    ''' based on the train & test set, Logistic Regression is the winner '''
    A = em.read_csv_metadata('A.csv', key='business_id')
    B = em.read_csv_metadata('B.csv', key='id')
    C = em.read_csv_metadata('C.csv', key='_id',ltable=A, rtable=B,
fk_ltable='ltable_business_id', fk_rtable='rtable_id')
    match_f = mm.get_feats(A, B)
    K = em.extract_feature_vecs(C, feature_table=match_f,
                                 attrs_before= ['_id', 'ltable_business_id',
'rtable_id'])
    K = K.fillna(1)
    attrs_to_exclude = ['_id', 'ltable_business_id', 'rtable_id']
    G = em.read_csv_metadata('G.csv', key='_id', ltable=A, rtable=B,
fk_ltable='ltable_business_id', fk_rtable='rtable_id')
    Gfvs = mm.train_fvs(G, match_f)
    Gfvs = Gfvs.fillna(1)

    #em.to_csv_metadata(K, './K.csv')
    K = em.read_csv_metadata('K.csv', key='_id', ltable=A, rtable=B,
fk_ltable='ltable_business_id', fk_rtable='rtable_id')

    lg = em.LogRegMatcher(name='LogReg', random_state=77)
    lg.fit(table=Gfvs, exclude_attrs=attrs_to_exclude, target_attr='gold_labels')

    predictions = lg.predict(table=K, exclude_attrs=attrs_to_exclude,
              append=True, target_attr='predicted', inplace=False)

    matches = predictions.loc[predictions['predicted'] == 1]
    return matches

def matchesfvs_to_orig(matches):
    ''' output of get_all_matches() is in feature vector form
        merge with original csv files
    '''
    Y = em.read_csv_metadata('yelp_restaurants.csv', key='business_id')
    Z = em.read_csv_metadata('zomato_restaurants.csv', key='id')
    matches_y = matches[['ltable_business_id', 'rtable_id']].merge(Y,
left_on='ltable_business_id', right_on='business_id')
    matches_y_z = matches_y.merge(Z, left_on='rtable_id', right_on='id')
    em.to_csv_metadata(matches_y_z, './matches.csv')
```

```
def cleanup(M):
    '''
    Drop repetitive or unnecessary columns.
    For Zomato, replace full addresses with short addresses:
        the first part of an address before a comma,
        since we already have city, state, zipcode attributes.
    Take Yelp's city_x over Zomato's similar locality attribute.
    Take Yelp's postal_code over Zomato's (which are missing some zipcodes).
    '''
    # drop repetitive columns
    M.pop('business_id')
    M.pop('id')
    # drop columns which offer no information
    M.pop('state') # are all 'WI'
    M.pop('type') # are all 'business'
    M.pop('R') # same as zomato id
    M.pop('apikey')
    M.pop('url')
    M.pop('location') # already extracted to separate columns in jsontocsv.py
    M.pop('switch_to_order_menu') # all 0
    M.pop('currency') # all $
    M.pop('offers') # all []
    M.pop('thumb')
    M.pop('photos_url')
    M.pop('menu_url')
    M.pop('featured_image')
    M.pop('has_online_delivery') # all 0
    M.pop('is_delivering_now') # all 0
    M.pop('deeplink')
    M.pop('has_table_booking') # all 0
    M.pop('events_url')
    M.pop('establishment_types') # all []
    M.pop('city_y') # from Zomato, all Madison
    M.pop('city_id') # from Zomato, all Madison
    M.pop('country_id') # all 216 (probably U.S.)
    # locality_verbose is just locality + 'Madison'
    #M['locality_verbose'] = M['locality_verbose'].str.extract('^(.+?),')
    #M['locality_verbose'].equals(M['locality'])
    M.pop('locality_verbose')
    M['address_y'] = M['address_y'].str.extract('^(.+?),')
    #M['city_x'].value_counts()
    #M['locality'].value_counts()
    # take yelp's city_x over zomato's more finer locality
    M.pop('locality')
    #M['comp_zips'] = M.postal_code == M.zipcode
    # take yelp's postal_code over zomato's zipcode
    M.pop('zipcode')

    # rename for clarity
```

```
            M = M.rename(columns={'ltable_business_id': 'yelp_id', 'rtable_id':
'zomato_id', 'postal_code': 'zipcode', 'city_x': 'city'})

            return M

        '''
        None of the compute_* functions should perform side effects. See the docs for
        pandas.DataFrame.apply() for more information.
        '''

        def compute_name(row):
            '''
            Chooses the name with the longer length.
            '''
            if len(row['name_x']) < len(row['name_y']):
                row['name'] = row['name_y']
            else:
                row['name'] = row['name_x']
            return row

        def compute_address(row):
            '''
            Chooses the address with the longer length.
            '''
            if len(row['address_x']) < len(row['address_y']):
                row['address'] = row['address_y']
            else:
                row['address'] = row['address_x']
            return row

        def compute_latitude(row):
            '''
            Computes the latitude as the average (arithmetic mean) of latitude_x and
            and latitude_y. Assumes that none of the latitudes are 0.
            '''
            row['latitude'] = (row['latitude_x'] + row['latitude_y']) / 2
            return row

        def compute_longitude(row):
            '''
            Computes the longitude as the average (arithmetic mean) of longitude_x and
            longitude_y. Assumes that none of the longitudes are 0.
            '''
            row['longitude'] = (row['longitude_x'] + row['longitude_y']) / 2
            return row

        def main():
            matchesfvs_to_orig(get_all_matches())

            uncleanM = em.read_csv_metadata('./matches_cleaned.csv')
```

```
    M = cleanup(uncleanM)

    M['name'] = pd.Series()
    M = M.apply(compute_name, axis=1)
    M.pop('name_x')
    M.pop('name_y')

    M['address'] = pd.Series()
    M = M.apply(compute_address, axis=1)
    M.pop('address_x')
    M.pop('address_y')

    M['latitude'] = pd.Series()
    M = M.apply(compute_latitude, axis=1)
    M.pop('latitude_x')
    M.pop('latitude_y')

    M['longitude'] = pd.Series()
    M = M.apply(compute_longitude, axis=1)
    M.pop('longitude_x')
    M.pop('longitude_y')

    em.to_csv_metadata(M, './E.csv')

if __name__ == '__main__':
    main()
```