[CS 838 - Spring 2017] Stage 3 Report

Tarun Bansal, Ayush Gupta, Rohit Damkondwar

1. **Type of entity matched and description of tables.**

We have taken two datasets containing information about top restaurants in different cities across the world [yelp_data.csv and zomato_data.csv] . In this stage we tried to match tuples in both the datasets that refer to the same establishment in real world.

Number of Tuples in Table A (yelp) : 144072
Number of Tuples in Table B (zomato) : 3600

For convenience while matching we have created the csv files such that the have common attributes : ID, name, address, city, zipcode, latitude, longitude, review_count, rating, zomato_id, yelp_id

2. **Blocker used and the number of tuples obtained in the candidate set after blocking.**

For blocking potential similar tuple pairs and to increase the speed, first we performed hash based blocking based on the zipcode of restaurants. This ensures further blocking rules are applied only on restaurants which have the same **zip code**.

This step leaves us with **361,156 tuple pairs**, which is a huge number. So to narrow down our search of potential matches, we apply an **edit distance** based similarity rule on the **name of establishments**. All tuple pairs with similarity measure less than 0.4 are removed from the set of potential matching tuple pairs.

So after applying this final blocking rule we are left **2472 tuple pairs** which is a significant reduction.

3. **Number of tuple pairs in the labeled sample G.**

From the set of 2472 blocked tuple pairs we take a sample of **600 tuple pairs** and label them. This creates our labeled sample G.

4. **Cross validation results and classifier selection on Set I**

We split our labeled Sample G into Set I and Set J which have **300 tuple pairs each**. We intended to use the sample I for training and cross validation purposes. Set J which will be used for testing later was left untouched at this step.

We applied all the six learning methods (Decision Tree, Random Forest, SVM, Naive Bayes, Logistic Regression, Linear Regression) and report the precision, recall, and F-1 after performing cross validation for the first time for these methods on Set I :

ITERATION #1

Classifier	Precision	Recall	F-1
Decision Tree	0.917409	0.890635	0.901056
Random Forest	0.973333	0.900847	0.934535
SVM	0.958942	0.8376	0.892096
Logistic Regression	0.942716	0.88418	0.91214
Linear Regression	0.936564	0.879082	0.906632
Naive Bayes	0.952124	0.84051	0.88743

Since Random Forest has precision and recall measures slightly above 0.9 and the best F-1 measure (0.934535) we selected it as our learning based matcher.

5. **Problems encountered and how did we fix them, information about debugging iterations.**

Since we have already achieved very good precision and recall measures after our first cross validation, we didn't perform any more debugging iterations. We keep random forest as our matcher with following P, R and F-1 measures for each classifier after final iteration:

Classifier	Precision	Recall	F-1
Decision Tree	0.917409	0.890635	0.901056
Random Forest	0.973333	0.900847	0.934535
SVM	0.958942	0.8376	0.892096
Logistic Regression	0.942716	0.88418	0.91214
Linear Regression	0.936564	0.879082	0.906632
Naive Bayes	0.952124	0.84051	0.88743

Before performing cross validation, we ran into a problem of **skewed data** while the labelling step. Here due to very strict blocking rules we had very few negative examples in our labeled

data (around **10 out of 600**). Hence we decided to relax the edit distance similarity blocking rule from 0.8 to 0.4. This helped us in creating a more balanced labeled data having a reasonable number of negative examples.

6. **Final best matcher selected and its P/R/F-1 on Set I.**

After performing cross validation we settle on Random Forest as our best matcher on Set I. The Precision, Recall and F-1 measures are :

Classifier	Precision	Recall	F-1
Random Forest	0.973333	0.900847	0.934535

7. **Results of every classifier after training on I and testing on Set J.**

For each of the six learning methods, training the matcher based on that method on I, we report its precision/recall/F-1 on J as :

Classifier	Precision	Recall	F-1
Decision Tree	0.9355	0.8969	0.9158
Random Forest	0.9677	0.9278	0.9474
SVM	0.9355	0.8969	0.9158
Logistic Regression	0.9307	0.9691	0.9495
Linear Regression	0.9778	0.9072	0.9412
Naive Bayes	0.9674	0.9175	0.9418

8. **Results of best classifier obtained after testing on Set J.**

The best matcher we selected after training on I was Random Forest and the P,R,F-1 for Random Forest on Set J is :

Classifier	Precision	Recall	F-1
Random Forest	0.9677	0.9278	0.9474

9. **Time estimates to do the blocking, label the data, find the best matcher.**

These time estimates include time spent in reading Magellan Documentation and trying them for our datasets.
Time to do the blocking : 3 hours
Time to label the data : 3 hours
Time to train and test and hence find the best matcher : 6 hours

10. **Why we could not achieve higher recall and what can be done in future to improve recall.**

Recall for Random Forest is 0.92 for our data set. To further improve the recall, we can improve address based similarity score function. Also, some of the auto generated features can be discarded.

11. **[BONUS] Feedback on Magellan.**
    a. Clear and detailed documentation made entity matching with Magellan very easy.
    b. Slick API for almost every entity matching requirement.

Some suggestions to improve Magellan :

c. Provide options to read datasets from sources other than csv files. Eg. We had our data in json files.
d. Allow users to directly use their database connections to import data. For eg. import data from MySQL db.
e. While reading CSV data (**em.read_csv_metadata**) provide users an option to specify a **custom delimiter**. For eg in our case certain names of establishments had commas in it. We had to convert it to $*$ so as to retain them which is a big negative.
f. While selecting a matcher during cross validation (**em.select_matcher**) provide users option to **report multiple metrics** at once. For eg :

```
metric = ['precision','recall','f-1']  #Currently not supported
```

g. Ease the setup process for Python 2 and Windows users. Not possible to debug and visualize using this combination as PyQT requires python 3 for windows.
h. PyQT seems to be optional dependency. Since, PyQT installation is painful, can it be replaced by some python platform independent package (Similar to Java Swing)?
i. Speed up blocking by providing advanced users an option to use distributed computing.