CS 838 (Spring 2017): Data Science
Project Report - Stage 3 (Group 12)

Deepanker Aggarwal
deepanker@cs.wisc.edu

Saket Saurabh
ssaurabh@cs.wisc.edu

Vishnu Lokhande
lokhande@cs.wisc.edu

1 Objective

In an earlier project stage, we had collected two structured datasets from different sources. In this stage of project, we do **entity matching** over these two datasets. The goal is to match tuples (rows of tables) from the first table to the second table based on the criterion that they describe the same entity.

We use Magellan , an entity matching tool, for this project stage.

2 Procedure

First, we need to extract data from the two sources in structured form such that they have the same schema. Let the two extracted tables be A and B. As the tables A and B are two large to work with, we downsample them into smaller tables, let us call them, $sample\_A$ and $sample\_B$. We need to match tuples from $sample\_A$ to those from $sample\_B$, hence we need to work on a table which contains the set of all tuple pairs from $sample\_A$ and $sample\_B$. Now, we introduce of stage called *Blocking*, which filters out the most matching tuple pairs. Let this table be C which stands for candidate set. We work on a sample of C which is called S and then manually label S such as match or no-match for each tuple pair to get a table named G. We extract features from G and learn classifiers on G. Figure.2 illustrates this procedure.

3 Data Description and Downsampling

The first structured data is the Yelp restaurant dataset which contains the information of different restaurant businesses in New York City.
The second structured data is the New York City Restaurant Inspection Dataset

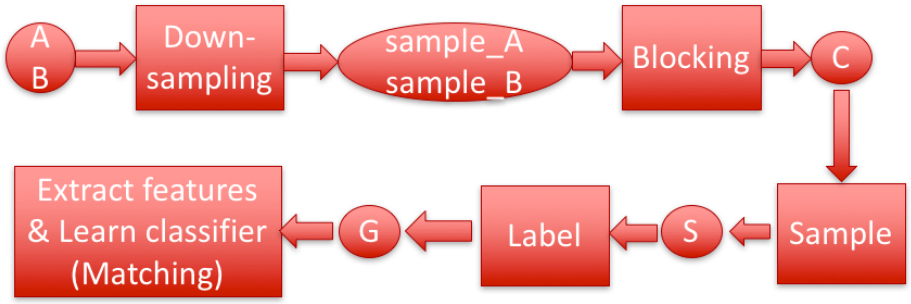

Figure 1: Entity Matching flowchart

which contains inspection results for various NYC restaurants that are conducted by New York City Department of Health and Mental Hygiene (DOHMH). For more details on the dataset, please refer Project Stage 1.

- The entity which we are trying to match are **restuarants**.
- The number of tuples in the first dataset, table A, are **19270**.
- The number of tuples in the second dataset, table B, are **23907**.

After downsampling,

- The number of tuples in $sample_A$ are **15635**.
- The number of tuples in $sample_B$ are **18000**.

4 Blocking and Labelling

The attributes in the two tables A and B are *name, address, zipcode* and *cuisine*. The size of the set containing tuple pairs before blocking is $15635 \times 18000 = 281430000$.
For blocking, we first do attribute equivalence blockers on *zipcode* and then we do overlap blocker on *address*.

- The number of tuples in the candidate set are **1559**

We then sample 1000 tuple pairs from the candidate set for labelling.

- The number of tuples in the labelled set, G, are **1000**

5 Feature Extraction and Matching stage

We divide the labeled set G, into set I and set J. We develop algorithms on set I and use set J to report accuracies. The cross-validation scores are reported

on set I. During the debugging stage, set I has been divided into set P (for training) and set Q (for testing), in order to identify the false positives and false negatives.

5.1 First iteration

5.1.1 Features Used

We used 3 features on name described below:

- Jaccard similarity measure on restaurant name
- Overlap similarity measure on restaurant name
- Exact match on restaurant name

5.1.2 Cross-validation

- Cross Validation results for the first time

Matcher	Precision(Mean)	Recall(Mean)	F1(Mean)
Decision Tree	0.954497	0.898707	0.925463
Naive Bayes	0.940495	0.932362	0.936340
Random Forest	0.954768	0.893945	0.922586
SVM	0.940495	0.932362	0.936340
Linear Regression	0.945024	0.932362	0.938578
Logistic Regression	0.948239	0.932362	0.940171

- Matcher Chosen
  Decision Tree as highest Precision and good enough recall. Target is to increase recall

5.2 Second iteration

5.2.1 Debugging stage

- Matcher to debug
  Decision Tree
- Problems found and fix

  Address was an exact match(ignoring punctuation marks) but the name was a prefix in some cases and a lot of times the restaurant changed its name or has a completely different name for the same address. eg: Yelp had restaurant name as cloud 28 whereas health inspection data had name as stix chelsea/hotel indogi/rooftop bar We decided to remove the Overlap similarity measure on restaurant name as it was lowering the recall because of the way names were written, and added 3 new features on restaurant's address mentioned below:-

- Custom similarity measure on address which looked for normalized amount of street number and block number matches in the address and jaccard measure on street name
- Overlap similarity measure on restaurant address
- Exact match on restaurant address

- Accuracy
  Before Iteration:
  Decision Tree

Precision	Recall	F1
0.9667	0.9206	0.9431

  After Iteration:
  Decision Tree

Precision	Recall	F1
0.959	0.9286	0.9435

5.2.2 Cross-validation

- Cross Validation results after initial debugging iterations

Matcher	Precision(Mean)	Recall(Mean)	F1(Mean)
Decision Tree	0.958756	0.910801	0.933709
Naive Bayes	0.961032	0.923208	0.941250
Random Forest	0.959165	0.910295	0.933896
SVM	0.965419	0.932362	0.948444
Linear Regression	0.944814	0.940514	0.942379
Logistic Regression	0.955828	0.932362	0.943808

- Matcher Chosen
  SVM as highest Precision and very high recall.

5.3 Third iteration

5.3.1 Debugging stage

- Matcher to debug
  SVM

- Problems found and fix

  In an attempt to increase the recall, we noticed that there were false negatives where address was a match, name wasn't an exact match but cuisine was a match. So we decided to add feature to match cuisine using overlap similarity measure.

- Accuracy
  Before Iteration:
  SVM

Precision	Recall	F1
0.9832	0.9286	0.9551

  After Iteration:
  SVM

Precision	Recall	F1
0.9833	0.9365	0.9593

5.3.2 Cross-validation

- Cross Validation results after initial debugging iterations

Matcher	Precision(Mean)	Recall(Mean)	F1(Mean)
Decision Tree	0.962461	0.909429	0.935093
Naive Bayes	0.961032	0.927290	0.943399
Random Forest	0.973125	0.927099	0.949491
SVM	0.964821	0.939333	0.951533
Linear Regression	0.952322	0.940514	0.946028
Logistic Regression	0.964821	0.935251	0.949384

- Matcher Chosen
  SVM as very high Precision and recall

Final Best Matcher chosen by us was Linear Regression with results on J set as:-

Precision	Recall	F1
0.9567	0.9404	0.9485

6 Final Results for each matcher on Set J

Matcher	Precision(Mean)	Recall(Mean)	F1(Mean)
Decision Tree	0.100	0.9489	0.9738
Naive Bayes	0.9648	0.9319	0.9481
Random Forest	0.9955	0.9489	0.9717
SVM	0.9651	0.9404	0.9526
Linear Regression	0.9567	0.9404	0.9485
Logistic Regression	0.9649	0.9362	0.9503

7 Approximate Time estimates

The time estimates for different stages are as follows-

- Blocking
  Blocking code takes around 52 seconds to execute.
- Labelling data
  We had to manually label the data. It took around 3 hours for us to label the data.
- Finding the best matcher
  The debugging and feature extraction stage took us around 5 hours. The code for matching stage execute in less than a second.

8 Magellan review - Bonus Points

Here are some of the good things about using Megallan.

- Magellan gives very good API's for performing all the necessary tasks for entity matching.
- The automated features functionality of Magallan is very useful for a quick build of the entity matching pipeline. It's performance is almost at par with the manual features that we coded up.

Here are somethings which can be improved in Megallan.

- Seed functionality should be provided for the down_sample API and the sample_table API. This helps us in working on the same sample set every time we run the code
- Installation instruction should have been provided by the Magellan team for PyQT4 software as PyQT4 doesn't have user-friendly installation instructions.
- There were many mismatches, in the keywords used in the syntax of an API, between the user manual of Magellan and the actual keyword asked for in the code.
- One possible future direction for Magellan can be to transform the entire pipeline as a GUI. The user can just manually choose from the GUI what he intends to do instead of writing code.