CS839 Data Science Spring 2018

<Group 9>

Ya-Chun Yang, yang364@wisc.edu
Han Wang, hwang729@wisc.edu
Yanghui Kang, ykang38@wisc.edu

Stage 3 Report - Entity Matching

INTRODUCTION

In an earlier project stage, we had collected two structured datasets from two popular crowd-sourced review website: Yelp and Tripadvisor. In this stage project, we do entity matching over these two datasets. The goal is to match tuples (row 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.

DATA DESCRIPTION

The entity we are trying to match is "Restaurant" in Los Angeles, CA. We extracted two tables: Table A and Table B. Both tables share the following schema: name, category_1, category_2, address, city, zipcode, phone, price, rating, review_count, a set of attributes for open hours for both of them.

- Table A: extracted from Yelp, contains 3188 restaurants/tuples
- Table B: extracted from TripAdvisor, contains 3120 restaurants/tuples

BLOCKING AND LABELING

The size of the set containing tuple pairs before blocking is $3188 \times 3120 = 9946560$.

We use the following blockers to get the candidates:

1. AttrEquivalenceBlocker on zipcode
2. OverlapBlocker on phone: share at least 4 3-gram words
3. OverlapBlocker on address: share at least 2 3-gram words
4. OverlapBlocker on name: share at least 2 3-gram words

The number of tuples in the final candidate set after the blocking is 1745.

We labeled 500 tuples in the sample G.

MATCHING

We divide the label 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.

I. First Iteration

Table 1. Cross Validation results for the first time

Matcher	Precision	Recall	F1
Decision Tree	0.946289	0.962761	0.953599
Random Forest	0.969858	0.979259	0.974203
SVM	0.895076	0.942678	0.915440
Naive Bayes	0.985185	0.951852	0.966864
Logistic Regression	0.960386	0.949428	0.953793
Linear Regression	0.977000	0.971852	0.974205

We choose Naive Bayes as our best matcher because it has highest precision and good enough recall.

II. Debugging Iteration

We choose the best matcher, Naive Bayes, to debug. In an attempt to increase the recall, we noticed that the new feature set with extra feature (price + rating) could improve the precision of Naive Bayes matcher. So we decided to use this new feature set instead of the original one. The final Precision/Recall/F-1 is,

Precision	Recall	F1
1.000	0.8923	0.9431

III. Cross Validation Iteration

Table 2. Cross Validation results after the debugging iteration

Matcher	Precision	Recall	F1
Decision Tree	0.960696	0.962761	0.961258
Random Forest	0.976623	0.963502	0.969706
SVM	0.893651	0.933587	0.910848
Naive Bayes	0.985185	0.951852	0.966864
Logistic Regression	0.960386	0.949428	0.953793
Linear Regression	0.977000	0.971852	0.974205

We choose Naive Bayes as the final best matcher because it has highest precision, recall, and F-1.

IV. Final Results

Table 3. Final results for each matcher on Set J

Matcher	Precision	Recall	F1
Decision Tree	0.9118	0.9538	0.9323
Random Forest	0.9453	0.9308	0.9380
SVM	0.8832	0.9308	0.9064
Naive Bayes	0.9760	0.9385	0.9569
Logistic Regression	0.9600	0.9231	0.9412
Linear Regression	0.9764	0.9538	0.9650

The final best matcher (Naive Bayes) has results on J:

Precision	Recall	F1
0.9764	0.9538	0.9650

TIME ESTIMATES

Process	Work time
Blocking	2 hrs
Labeling	1 hr
Cross Validation	1 hr
Debugging	2 - 3 hrs

The blocking step took about 2 hrs including initial try and debugging. The labeling process requires about 1 hr.

It approximately took 1 hr to test the 6 classifiers in Magellan using cross validation to find the best matcher. We then tried to debug by putting some additional features in our feature set. We tested several features which are related to rating, price, and their combinations. We finally generated a feature set and the corresponding best matcher. The latter process took approximately 2-3 hr.

DISCUSSION

Improve the matcher accuracy

Our final result is good, the precision is 97.8% and recall is 95.4%, which means we filtered most of dirty data on the blocking process, and the feature set in matching process is distinguishable. The remaining false positive and negative examples are mostly induced by dirty data. Therefore, to further improve the accuracy, we need additional data cleaning processes on table A and B. More specifically, missing data is very common in address, phone number and zip code, and restaurant name and address might include symbols and spelling errors. Using more

powerful tools like deep learning might also help with missing data and errors. In additional, we could also generate more features and test the different combination of those features to produce a better feature set. We totally used 49 features in our matcher. It is possible to do feature selection to select a smaller size of feature set for efficiency manner while keeping a reasonable good accuracy.

Feedback on Magellan

Magellan is a great tool for entity matching. It not only offers a convenient interface for blocking, matching and feature generation process, but nicely interacts Pandas dataframe and several machine learning classifiers. In our data, the result is pretty good, indicating the amazing performance of Magellan.

One thing could be improve in Magellan is the installation step using conda . Although it automatically install the py_entitymatching and most of its dependencies via conda command, we could not import py_entitymatching correctly. The problem is Mac OS's compiler have switched from using GCC's libstdc++ as the default standard library to clang's libc++, but Anaconda is built with a much older compiler toolchain that still uses libstdc++. The two standard libraries are not fully compatible. As a result, we need to invoke the compiler to explicitly use the old C++ standard library under Anaconda in order to build the py_entitymatching, so that we are able to import the py_entitymatching module without any errors.

Another suggestion is on the treatment of missing values. In Magellan, missing value is not allowed in feature vectors. In this project, we employed the simplest way to impute feature vectors with the column means. However, such method might not work for all types of features or attributes. For text attributes (such as restaurant address and name) and related similarity measurement features (such as jacquard), imputing with column mean can be misleading and negatively affect the performance of classifier. Therefore, we suggest to add missing-value imputation functionality for the attributes as well as in the "feature generation" method in Magellan. The user might opt to impute missing-value with a selection of methods directly over the attributes before generating features, or ask the feature generator to take care of the missing

values. The feature generator can be “smart” in treating missing value by considering the type of data, distribution of data, as well as relationships with other attributes.