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

Group : 17
Gautam Singh
Harsh Singhal
Naman Agrawal

Objective : Entity matching (EM) using supervised learning techniques

In this project stage, we find entities across two datasets that match using supervised machine learning techniques. We use an EM tool named Magellan to match two tables having same schema and each tuple in both table describing entity type restaurant.

1. **Describe the type of entity you want to match, briefly describe the two tables (e.g., where did you obtain these tables), list the number of tuples per table.**

We are looking to match restaurant entity. We created two tables using data obtained by crawling Yelp and Zomato website and transforming obtained data to structured format. Table A (yelp.csv) which has 3075 tuples corresponds to data obtained from Yelp, whereas table B(zomato.csv) has 3478 tuples and corresponds to data obtained from Zomato website. Both tables have same schema and duplicate entries were removed from both tables during data cleaning. Each tuple in both table describe entity type restaurant with attributes -

< ID, Name, Phone, Zipcode, State, City, Address, Delivery, Takeout, Outdoor_seating >

We want to match the entities in two tables if they refer to the same restaurant based on values of attributes in both table.

2. Describe the blocker that you use and list the number of tuple pairs in the candidate set obtained after the blocking step.

Before carrying out blocking, we down-sample data from both the tables using Magellan.

(a) First we carry out blocking on "Address" using "overlap" similarity measure. The Address is tokenized using qgram as 3. To pass the blocking step, the addresses must have at least 3 same tokens.
(b) Then, we block the resulting tuples on "Name" using "overlap" similarity measure. The name is tokenized using qgram as 3. To pass the blocking step, the names must have at least 1 matching token.
(c) After that, the resultant tuples are subjected to "Attribute Matching" on "Zipcode".
To pass this blocking step, the zipcodes should exactly match.
(d) After blocking is done, we get 1688 tuples. We randomly down-sample it to 1100 tuples using magellan.
(e) We tagged the resultant table "to_be_tagged.csv" by adding a column "Label" and labelled "1 - match" and "0 - no match".

We obtained 1700 tuple pairs in the candidate set after the blocking step.

3. List the number of tuple pairs in the sample G that you have labeled.

We sampled 1100 tuples from candidate set to be manually labelled.

4. For each of the six learning methods provided in Magellan (Decision Tree, Random Forest, SVM, Naive Bayes, Logistic Regression, Linear Regression), report the precision, recall, and F-1 that you obtain when you perform cross validation for the first time for these methods on I.

We sampled 1000 tuples from manually labelled tuples and divided it into Train set I consisting of 700 tuples and Test set J consisting of 300 tuples. We performed cross validation using set I and obtained following Precision, Recall and F-1 values.

Model	Precision	Recall	F-1
Decision Tree	0.9824	0.9944	0.9882
Random Forest	0.9944	0.9944	0.9944
SVM	0.9947	0.9201	0.9555
Naive Bayes	0.9944	0.9851	0.9897
Logistic Regression	0.9863	0.9950	0.9904
Linear Regression	1.0000	0.9630	0.9811

5. Report which learning based matcher you selected after that cross validation.

We selected random forest based matcher as the best learning model after the first round of cross validation based on precision, recall and F-1 scores.

6. Report all debugging iterations and cross validation iterations that you performed. For each debugging iteration, report (a) what is the matcher that you are trying to debug, and its precision/recall/F-1, (b) what kind of problems you found, and what you did to fix them, (c) the final precision/recall/F-1 that you reached. For each cross validation iteration, report (a) what matchers were you trying to evaluate using the cross validation, and (b) precision/recall/F-1 of those.

We performed multiple iterations and found relatively higher precision, and recall values with random forest based matcher as compared to other models. As we received precision, recall and f1 values above 95 percent there was very little scope for improvement after debugging. Since, we did not change matching process, precision, recall and f-1 values for all six models are same reported above(5) .

Model	Precision	Recall	F-1
Decision Tree	0.9824	0.9944	0.9882
Random Forest	0.9944	0.9944	0.9944
SVM	0.9947	0.9201	0.9555
Naive Bayes	0.9944	0.9851	0.9897
Logistic Regression	0.9863	0.9950	0.9904
Linear Regression	1.0000	0.9630	0.9811

7. Report the final best matcher that you selected, and its precision/recall/F-1.

Since, we did not change anything during debugging process, the final learning model we select is still random forest (RF). Precision/recall/F1 score of RF after splitting development set I into set A(50 %) and set B(50%) is as given below.

Model	Precision	Recall	F-1
Random Forest	1.000	0.9875	0.9937

False positives : 0 (out of 79 positive predictions)

False negatives : 1 (out of 229 negative predictions)

8. For each of the six learning methods, train the matcher based on that method on I, then report its precision / recall / F1 on J.

I. Decision Tree Learning
    Precision : 100.0% (77/77)
    Recall : 98.72% (77/78)
    F1 : 99.35%
    False positives : 0 (out of 77 positive predictions)
    False negatives : 1 (out of 253 negative predictions)
II. Random Forest Learning
    Precision : 100.0% (77/77)
    Recall : 98.72% (77/78)
    F1 : 99.35%
    False positives : 0 (out of 77 positive predictions)
    False negatives : 1 (out of 253 negative predictions)
III. SVM Learning
    Precision : 100.0% (74/74)
    Recall : 94.87% (74/78)
    F1 : 97.37%
    False positives : 0 (out of 74 positive predictions)
    False negatives : 4 (out of 256 negative predictions)
IV. Naive Bayes Learning
    Precision : 100.0% (77/77)
    Recall : 98.72% (77/78)
    F1 : 99.35%
    False positives : 0 (out of 77 positive predictions)
    False negatives : 1 (out of 253 negative predictions)
V. Logistic Regression Learning
    Precision : 100.0% (77/77)
    Recall : 98.72% (77/78)
    F1 : 99.35%
    False positives : 0 (out of 77 positive predictions)
    False negatives : 1 (out of 253 negative predictions)
VI. Linear Regression Learning
    Precision : 100.0% (73/73)
    Recall : 93.59% (73/78)
    F1 : 96.69%
    False positives : 0 (out of 73 positive predictions)
    False negatives : 5 (out of 257 negative predictions)

9. For the final best matcher Y selected, train it on I, then report its precision/recall/F-1 on J.

Precision : 100.0% (77/77)
Recall : 98.72% (77/78)
F1 : 99.35%
False positives : 0 (out of 77 positive predictions)
False negatives : 1 (out of 253 negative predictions)

10. Report approximate time estimates: (a) to do the blocking, (b) to label the data, (c) to find the best matcher.

11. Provide a discussion on why you didn't reach higher recall, and what you can do in the future to obtain higher recall.

12. Provide comments on what is good with Magellan and what is bad, that is, as users, what else would you like to see in Magellan. Are there any features/capabilities that you would really like to see being added? Any bugs?

Strengths:-

(a)Great documentation and easy to learn.
(b)Complete entity matching solution that combines various methods from various libraries like pandas, numpy, scikit-learn, etc.

Bugs :-

(a)In the method "block_candset", in the class "OverlapBlocker", the "show_progress=False" feature does not work. Even after specifying this, it always showed the progress in the output which wasn't desired.

(b)We noticed many a times that when reading the csv file, in the first run, the interpreter used to fire the error 'No handlers could be found for logger "py_entitymatching.io.parsers"' but used to work fine upon second run. Its requires re-running the code again and should be fixed.

Suggestions:-

(a)The down_sample() command carries out down sampling inconsistently i.e. each time, the resultant data set turns out to be a different set of tuples. We understand that this is how it is supposed to work, but there can be a scenario, where one might want to generate the same dataset again as it was generated in the last run. We came across such scenario in "CS532 Pattern Recognition" class where we came across a function in MATLAB that produces random numbers but the sequence of random numbers generated are consistent within unless it is reset in one lifecycle (till the program is in memory / shell is alive).
So, there should be a parameter available with this command which when specified should generate random samples in the first run but should give return the same set unless the memory is flushed or a command to unset the corresponding "flag" is explicitly specified by the user.
Please email us to further discuss it.

(b)This is an issue that is not exactly with magellan but we encountered it while writing the code.
In ipython, when we run individual sections of code, the result gets saved in the memory.
If we make a change in the variable name and run the code again, it will store the data across the new variable name.
However, while working, if we accidentally use the old variable name, then the interpreter will the old value which is still there in the memory. Hence, we will not get to know that we are using incorrect variable name and might give us deceiving / incorrect results. Hence, as a solution, they always verify our results by clearing the output and re-running the program to avoid such cases.