Part 4: Matching

Clarence Cheung {kcheung6@wisc.edu}
Jin Ruan {jruan3@wisc.edu}

Pre-Matching / Pre-processing

• Include additional info in Table C

We include ***streetAddress*** in Table C to enhance accuracy and efficiency in creating the Golden Table G.
(Please refer to the code in `block_and_sample.ipynb`)

• Further blocking

We need to do further blocking on the current candidate set (Table C) because the odds of selecting a true positive instance is roughly $\frac{3000}{430000} = 0.697\%$, which is extremely low. The following additional blocking rules are considered:

1. Do Overlapping (with $qgram = 5$) on ***name***
2. Do further Overlapping (with $overlap\_size$=2 and $qgram$= 5) on ***streetAddress***

The resulting candidate set C has around 21000 "matching" tuples, and so the odds of selecting a true positive instance increase to roughly 14% .
(Please refer to the code in `block_and_sample.ipynb`)

• Sampling

We randomly select 400 "matching" tuples using `Magellan`. There are 87 positive instances and 313 negatives. After that, we split 65% of the tuples for training (Table I) and the rest for testing (Table J.)

Selecting the best Machine-Learning matcher

Our plan to select the best matcher begins with using the Machine-Learning methods provided in `Magellan`:

1. Decision Tree (DT)
2. Random Forest (RF)
3. Support Vector Machine (SVM)
4. Naïve Bayes (NB)
5. Logistic Regression (LG)
6. Linear Regression (LN)

Then we perform cross validation (CV) on each of these methods with some features.

• **Feature Selection**

We decide to select features that are related to the available attributes from the candidate set (Table C.) In `Magellan`, a feature table is generated based on the existing schema and we can choose the existing features that are provided, or we can create our own features based on the attributes from the candidate set. The choices included are ***name***, ***streetAddress***, ***city***, ***state*** and ***zipCode***. Other remaining features are not considered because there are many NaN values and may affect the training process.

Iteration 1:

The table below shows the cross validation results for the first time:

Learner	Precision	Recall	F1
DT	0.921429	0.858009	0.886396
RF	0.983333	0.908009	0.940476
SVM	1.000000	0.785390	0.876366
NB	0.730702	1.000000	0.841852
LG	0.986667	0.888095	0.930966
LN	0.975000	0.929762	0.950920

After the first round cross validation, we selected SVM learner. Our criteria for selecting the learner is to consider the highest precision rate. Since SVM has an excellent precision rate of 1.0, it is selected in this iteration.

• **Debugging:**

Since the recall rate of SVM is low (0.785390,) debugging is necessary. However, `Magellan` does not have an built-in debugger for SVM and so we perform debugging using DT and RF. Also, we have written a python script that can show the mis-matched tuples and used that script for debugging. To do debugging, we split the training data I into smaller subsets of training set U and a testing set V. The result for precision, recall, F1 on V is shown below:

	Precision	Recall	F1
SVM	1.0000	0.7600	0.8636

• **Problem and Solution:**

There are many false negatives that share different names, but in fact the tuples refer to the same restaurant because their phone numbers are exactly the same while sharing very similar street address. Our solution is to add a phone matching feature (`exact_match(phone_phone)`) to the feature table.

Iteration 2:

After adding the phone matching feature, we perform cross validation again to verify if SVM is still the best matcher. The results are summarized in the following table:

Learner	Precision	Recall	F1
DT	0.971429	0.904762	0.934161
RF	1.000000	0.981818	0.990476
SVM	0.971429	0.946429	0.958095
NB	0.887698	1.000000	0.937667
LG	0.971429	0.946429	0.958095
LN	0.931044	0.946429	0.938429

Clearly RF is the best matcher in this iteration as it has an 100% precision rate and a very high recall rate of 98%.

• **Debugging:**

Although the cross validation precision and recall of RF are very high, we still want to know whether we can perform further improvement or discover any problem. Thus, we started to debug Random Forest in the same manner as the previous debugging process by splitting I into U and V. The precision, recall, F1 on V is shown below:

	Precision	Recall	F1
RF	0.9600	0.9600	0.9600

• **Problem and Solution:**

There are two tuples that are mislabeled: a negative is mislabeled as positive and a positive is mislabeled as negative. We need to correct the label in Table I and read again the corrected table.

Iteration 3:

After reading in the corrected tables, we examine if there are any significant differences on the "accuracy" statistics using the original set of features (without matching phone numbers.) Hence, we do cross validation on the six learners again and the results are summarized below:

Learner	Precision	Recall	F1
DT	0.904762	0.854762	0.877949
RF	0.983333	0.883333	0.923846
SVM	1.000000	0.763961	0.863179
NB	0.696732	1.000000	0.819725
LG	0.986667	0.866667	0.917779
LN	0.975000	0.908333	0.937733

Same as the first iteration, SVM is the best matcher with the highest precision rate with a decreased recall rate.

• **Debugging:**

Same debugging methods as mentioned in the first iteration.

	Precision	Recall	F1
SVM	1.0000	0.7600	0.8636

Same results as of the first iteration.

• **Problem and Solution:**

Same problem as experienced in the first iteration. Add phone number matching features as we did in iteration 2.

Iteration 4:

After adding the phone matching feature, we perform cross validation again to verify if SVM is still the best matcher. The results are summarized in the following table:

Learner	Precision	Recall	F1
DT	0.984615	0.935714	0.957926
RF	1.000000	0.975000	0.986667
SVM	1.000000	0.950000	0.973333
NB	0.880000	1.000000	0.932466
LG	1.00000	0.950000	0.973333
LN	0.959615	0.950000	0.953667

The numbers did not change and RF is still the best matcher.

- **Debugging:**

Same debugging procedure as iteration 2.

	Precision	Recall	F1
RF	0.9259	1.0000	0.9615

We decided not to further debug because those tuples have differed names and differed telephone and it is hard to debug. Otherwise,the numbers are promising enough to do testing on Table J.

Final matcher

We selected Random Forest as our best / final Machine-Learning matcher, and we are ready to do testing on J using this classifier.

- **Cross Validation on I:**

	Precision	Recall	F1
RF	0.981818	1.000000	0.990476

Manual rules

No manual rules are added to our best matcher.

Testing on J

Results of each classifier on Table J

Learner	Precision	Recall	F1
DT	0.9167	0.9429	0.9296
RF	0.9444	0.9714	0.9577
SVM	0.9429	0.9429	0.9429
NB	0.8500	0.9714	0.9067
LG	0.9444	0.9714	0.9577
LN	0.9167	0.9429	0.9296

We set Random Forest as our best matcher with the following statistics:

	Precision	Recall	F1
RF	0.9444	0.9714	0.9577

Time Spent

• Labeling

We spent less than 2 hours on labeling the Golden Table G, including identifying incorrect labels and relabel them correctly.

• Experiment with different trials

We spent more than 10 hours in experimenting different things to be used in the training process. These include further pre-processing the data, writing python utility scripts and experimenting any function available in `Magellan`.

• Finding the best learning-based matcher

Approximately 6 hours are spent on finding the best learning matcher. This involves trying different feature selections and adding rule based methods to attempt to improve precision and recall in some **preliminary training trials**. However, we did not add any rule in our final predictor.

• Debugging

We have probably spent 4 hours to debug various kinds of problems encountered in the training process.

Magellan Experience

1. It is a little bit hard to select disjointed features from the feature table, e.g. selecting feature[3:13], feature[15:17] and feature[20:24]. Users first need to select the desired features, then append each one to the final features structure (Data Frame.) It would be helpful if there is a function that can support such action in `Magellan`.

2. The documentation of using trigger is not comprehensive enough. It would be appreciated if more examples of utilizing triggers are provided.

3. On some occasions, the testing on J results for SVM are all zeros (0.00%) if multiple learning-based matchers are executed at the same time (in the same `ipython` cell,) and then printing them individually. Instead, we fix the problem by executing each learning-based matcher in each `ipython` cell.