CS838 DATA SCIENCE PROJECT

Entity Matching

Tuan Dinh
dinh5@wisc.edu

Sam Gelman
sgelman2@wisc.edu

Varun Sah
varun.sah@wisc.edu

April 2, 2017

CONTENTS

1 Data

The objective was to match electronics products belonging to several categories listed on Amazon and Newegg.

- From Amazon, we had data of 17518 products.
- From Newegg, we had data of 4160 products.

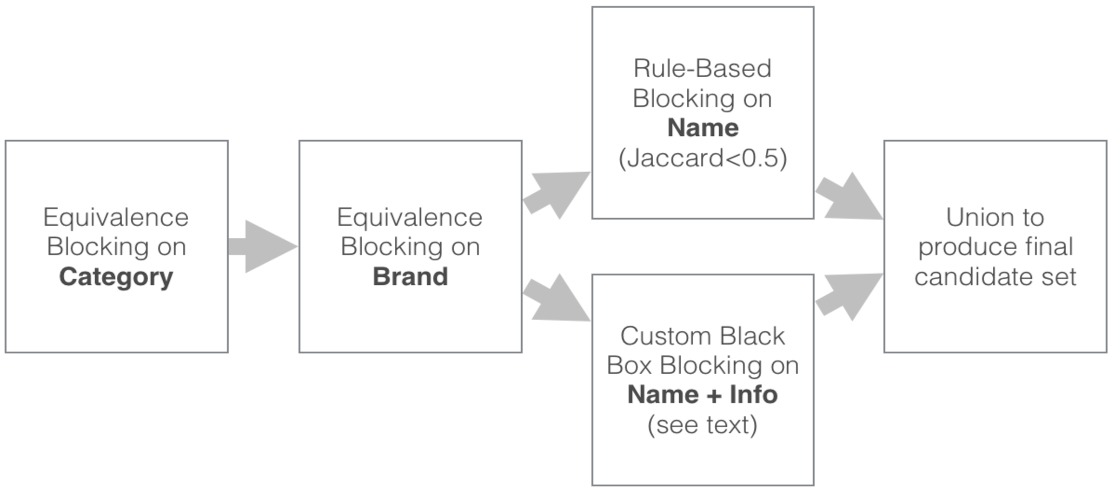
2 Data Cleaning and Schema Standardization

Data from the two sources was cleaned and standardized to have the same schema:

- ID: Product's ASIN on Amazon and NID on Newegg
- NAME: Product's title on both Amazon and Newegg
- CATEGORY: Product's bnode on Amazon and category on Newegg (both mapped to standardized strings specified in category_maps.csv)
- PRICE: Product's price on both Amazon and Newegg
- NUM_REVIEWS: Number of reviews on the product's pages on both Amazon and Newegg
- BRAND: Product's brand on both Amazon and Newegg
- INFO: Product's specifications on Amazon and model number on Newegg

3 Blocking and Labelling

The figure below shows an overview of our blocking process.


Equivalence Blocking on **Category**
Equivalence Blocking on **Brand**
Rule-Based Blocking on **Name** (Jaccard<0.5)
Custom Black Box Blocking on **Name + Info** (see text)
Union to produce final candidate set

The first block is an equivalence block on the category attribute, which contains standardized category names for each product. There are very few false positives in this step. The next block is an equivalence block on the brand attribute. We cleaned the data to make sure that most brand names were identical between the two datasets. The Magellan debugger was particularly useful for identifying brand names that needed adjusting. Some false positives slipped through, but we believe they had minimal impact on the final result. At this point, the blocking splits into two separate approaches that get combined to produce the final candidate set.

The first approach is a rule-based blocking on the name attribute. The Jaccard score between the two product names must be greater than 0.5 for the given tuple to end in up the final set. We chose Jaccard because it is normalized by the lengths of the names. Overlap was not a good pick since there are names as small as 4 words and as high as 15-20. The threshold of 0.5 was chosen after experimentation. We tested thresholds 0.2, 0.3, 0.4, and 0.6. We believe 0.5 gives the best balance between false positives and reducing the final size of the candidate set.

The second approach is a custom black box blocking on the name and info attributes. The Newegg info attribute contains the model number of the product. The model number is extremely useful for matching products between since it uniquely identifies a product. The Amazon info attribute contains several descriptions of the product. For many products, the model number is in one of those descriptions. When the model number is not in the Amazon info attribute, it is usually in the product name. Our custom black box blocker takes advantage of the preceding information. For a given tuple pair, it checks if the Newegg info attribute is in the Amazon info or name attribute. If so, the tuple pair passes the block. Combined with the rule-based blocking described in the previous paragraph, we believe we are getting precise blocking without too many false positives.

We took full advantage of the Magellan debugger at each step to verify our blocking results.

- Number of tuple pairs in the cartesian product: 72874880
- Number of candidate tuple pairs (after attribute equivalence on category): 3842626
- Number of candidate tuple pairs (after attribute equivalence on brand): 391609
- Number of candidate tuple pairs (after rule based blocking on name) : 3997
- Number of candidate tuple pairs (after custom blocking on info and name) : 1313
- Final number of candidate tuple pairs (union of rule based and custom blockers): 4938

We labelled a sample of 500 tuple pairs from the 4938 candidate tuple pairs. Of these, 170 tuple pairs were found to be actual matches.

4 Matching

4.1 Pre-processing

We started by loading the two datasets and the labelled tuple pairs after blocking. We then had to update the attribute type of 'PRICE' from string to numeric. We recognized the need for this after initial debugging on our training set, as Magellan considered the 'PRICE' of both Amazon and Newegg data as a string attribute. As a result, Magellan selected string matching or similarity methods for this attribute, which wasn't desirable. After the aforementioned pre-processing, we divided our dataset equally into training set I and test set J.

4.2 Feature Extraction

We created feature vectors by utilizing the feature generator provided by Magellan. The feature generator created a total of 26 features. We then extracted feature vectors for training set I and imputed missing values by median. After debugging, we added several of our custom features rather than using the features generated by Magellan.

4.3 Selecting the best classifier

To choose the most appropriate classifier, we ran 5-fold cross validation on the following models:

- Decision Tree
- Random Forest
- SVM
- Naive Bayes
- Logistic Regression
- Linear Regression

4.3.1 Selecting initial classifier

Classifier	*Precision*	*Recall*	*F1*
Decision Tree	0.723492063492	0.725072463768	0.708469785575
Random Forest	0.750683760684	0.492642140468	0.583212121212
SVM	0.687459207459	0.532062430323	0.587254480287
Naive Bayes	0.604492753623	0.788361204013	0.65076105433
Logistic Regression	0.559696969697	0.21762541806	0.294466936572
Linear Regression	0.636742424242	0.410613154961	0.481662219267

We picked Random Forest as the classifier M based on highest precision. But, since Random Forest's precision value was not acceptable (≥ 90%), we had to debug our methodology.

4.3.2 Debugging and Cross Validation : Iteration 1

To improve precision, we consider false positives. We tried to debug the Random Forest classifier precision. We randomly split set I into P and Q. Set P was used to train the classifier and set Q to test it. Random Forest performance on set Q before debugging iteration 1:

Classifier	*Precision*	*Recall*	*F1*
Random Forest	57.14% (16/28)	35.56% (16/45)	43.84%

- Diagnostics:
  - The features did not compare prices
  - Magellan considered 'PRICE' as string type instead of numeric
  - Irrelevant features like customer reviews comparisons
  - Since category and brand name were already used to block, their values were similar for most tuple pairs. But our classifiers had too many features related to these two attributes which was increasing the probability of positive predictions.
  - Some examples were predicted as positive even though price_small values were 0.
  - Jaccard similarity values of ('INFO', 'INFO') were very high already after blocking. Maybe it was not a good feature because many product pairs shared a lot of similar tokens.
- Solution:
  - Added 1 feature to compare prices: check if price difference is less than 1
  - Explicitly cast atttribute types of 'PRICE' into numeric
  - Removed all features related to attributes 'NUM_REVIEWS', 'CATEGORY', 'BRAND' and 'INFO', but kept features related to attribute 'NAME'
  - Added 2 extra 'PRICE' comparisons: exact (<0.01) and too large (> 10)

Random Forest performance on set Q after debugging iteration 1:

Classifier	*Precision*	*Recall*	*F1*
Random Forest	80.0% (28/35)	62.22% (28/45)	70.0%

After applying the debugging solution for iteration 1, the cross validation results were as follows:

Classifier	*Precision*	*Recall*	*F1*
Decision Tree	0.742329721362	0.819108138239	0.765191597127
Random Forest	0.817581699346	0.745607580825	0.762606606607
SVM	0.590649350649	0.32508361204	0.372794485455
Naive Bayes	0.391692307692	0.916666666667	0.527160486393
Logistic Regression	0.59	0.255596432553	0.32294453305
Linear Regression	0.658095238095	0.255596432553	0.487009222661

Random Forest was still the best classifier but its precision was not acceptable. Hence, we continued to debug.

4.3.3 Debugging and Cross Validation : Iteration 2

To improve precision, we consider false positives. We tried to debug the Random Forest classifier precision. We randomly split set I into P and Q. Set P was used to train the classifier and set Q to test it.
Random Forest performance on set Q before debugging iteration 2:

Classifier	*Precision*	*Recall*	*F1*
Random Forest	80.0% (28/35)	62.22% (28/45)	70.0%

- Diagnostics:
  - There were some wrong matches between 'normal' product and 'refurbished' products. 'refurbished' was the key word to eliminate incorrect positive predictions
  - 'INFO' of Newegg data has usually 1 token, which is the model number, and this is included in 'INFO' or 'NAME' of Amazon data
- Solution:
  - Added a feature 'refurbisfed' to check if 2 products are different
  - Added a feature 'model in name' to check if model number is included.

Random Forest performance on set Q after debugging iteration 2:

Classifier	*Precision*	*Recall*	*F1*
Random Forest	90.0% (36/40)	80.0% (36/45)	84.71%

After applying the debugging solution for iteration 2, the cross validation results were as follows:

Classifier	*Precision*	*Recall*	*F1*
Decision Tree	0.851282051282	0.861884057971	0.849029138503
Random Forest	0.868452380952	0.850579710145	0.854742830605
SVM	0.97032967033	0.747079152731	0.840472312002
Naive Bayes	0.398285714286	0.933333333333	0.53823740947
Logistic Regression	0.97032967033	0.747079152731	0.840472312002
Linear Regression	0.97032967033	0.747079152731	0.840472312002

Now, the best matcher was SVM. Its precision (97.03 %) was acceptable ($\geq 90\%$). Hence, we proceeded to train the SVM classifier on set I and test it on set J.

4.4 Testing the classifiers

We trained each of the six classifiers on set I and tested them on set J. The metrics for the test set performance are as follows:

- DecisionTree
  - Precision : 92.21% (71/77)
  - Recall : 81.61% (71/87)
  - F1 : 86.59%
  - False positives : 6 (out of 77 positive predictions)
  - False negatives : 16 (out of 173 negative predictions)
- RandomForest
  - Precision : 89.16% (74/83)
  - Recall : 85.06% (74/87)
  - F1 : 87.06%
  - False positives : 9 (out of 83 positive predictions)
  - False negatives : 13 (out of 167 negative predictions)
- SVM
  - Precision : 95.65% (66/69)
  - Recall : 75.86% (66/87)
  - F1 : 84.62%
  - False positives : 3 (out of 69 positive predictions)
  - False negatives : 21 (out of 181 negative predictions)
- NaiveBayes
  - Precision : 33.74% (83/246)
  - Recall : 95.4% (83/87)
  - F1 : 49.85%
  - False positives : 163 (out of 246 positive predictions)
  - False negatives : 4 (out of 4 negative predictions)
- LogisticRegression
  - Precision : 95.65% (66/69)
  - Recall : 75.86% (66/87)
  - F1 : 84.62%
  - False positives : 3 (out of 69 positive predictions)

    - False negatives : 21 (out of 181 negative predictions)

- LinearRegression
    - Precision : 95.65% (66/69)
    - Recall : 75.86% (66/87)
    - F1 : 84.62%
    - False positives : 3 (out of 69 positive predictions)
    - False negatives : 21 (out of 181 negative predictions)

Final Best Matcher:

Classifier	*Precision*	*Recall*	*F1*	*FP*	*FN*
SVM	95.65% (66/69)	75.86% (66/87)	84.62%	3(out of 69)	21 (out of 181)

The precision (95.65%) of our SVM classifier on the test set was acceptable (≥ 90%) with a recall of 75.86%.

5 Time Statistics

- Execution time of blocking script: 111.317143 seconds
- Execution time of matching scripts: 9.450438 seconds
- Execution time of cross validation: 0.3 seconds
- Time of coding - cleaning, blocking and debugging: 10 hours
- Time of labeling: 4 hours
- Time of coding - matching and debugging: 20 hours

6 Discussion

Current recall value is 76%. To obtain higher recall, we need to reduce our false negatives. The reasons for this are discussed below:

- The root cause behind low recall is the inherent nature of data from both the data sources. Despite thorough cleaning and some standardization performed by us, there are still inconsistencies that exist between data from both sources.
- Another reason might be our notion of what constitutes a "match". We decided that 2 products matched if their model numbers were same. Due to this, some false negatives happened because:
    - Some products had almost everything being same but their model numbers have only 1 different character.

    - Their prices on Amazon and Newegg were really different (maybe noisy data)
    - One had ’refurbished’ in its name and the other one did not.

Potential solutions to tackle these issues colud be:

- More thorough cleaning and standardization of data.
- Cross checking product prices manually.
- Reducing confidence level of classifier since a high enough precision has been achieved.

-

7 Magellan : Feedback

7.1 What’s great

- Debugger: We were genuinely impressed by the Magellan debugger. It helped us develop a balanced blocking solution and even showed us some inconsistencies that needed to be cleaned up in the dataset. We would not have caught some of them if it hadn’t been for the debugger.
- Jupyter notebooks: We felt that the Jupyter notebooks included with the Magellan documentation were a great place to start.

7.2 Bugs and suggestions for improvements

- Rule-based blocking: We felt that the API for rule-based blocking is unintuitive which made it harder to use. We encountered a bug when using "em.get_feature_fn". The dictionary returned by that function wasn’t accepted by "em.add_feature". We had to modify the dictionary by manually setting the values of "right_attribute" and "left_attribute" to get it to work.
- Update table feature: We found that Magellan could not cast all the attribute type automatically. For example, it recognized "NUM_REVIEWS" as numeric it didn’t recognize "PRICE" (only 0 or real numbered value) as a numeric feature. As a result, we could not use the "em.get_features_for_matching" method directly. As a workaround, we had to obtain the attribute types ("em.get_attr_types") and update it, then block corresponding attributes ("em.get_attr_corres()"), then get a list of similarity functions and a list of tokenizing functions ("em.get_sim_funs_for_matching()" and "em.get_tokenizers_for_matching()" respectively) and pass them to the "em.get_features" function. It would have been great if there was an easier way to just change the attribute type of tables.
- Multiple cross validation metrics: We felt that Magellan should support measuring many metrics on cross validation simultaneously. It currently seems to support just one metric each time.

- Remove_feature option: We felt a "remove_feature" option would be really useful. Specially in cases where we want to reuse almost all automatic features generated by Magellen, the easiest way would be to just remove a few irrelevant features.
- Python version: Finally, it would be great if Magellan could be brought up to date with the latest python version.