CS 839 Project Stage 3:
Entity Matching - Matching Laptops from Online Retailers

Aishwarya Ganesan
ag@cs.wisc.edu

David Liang
david.liang@wisc.edu

Viswesh Periyasamy
vperiyasamy@wisc.edu

Entity Tuples

The entity type that we chose to match was laptops, specifically sold from the online retailers Amazon and Walmart. We obtained all tuples from both of these web sources from stage 2 of the project and created two tables with a matching schema. Each tuple contains the following attributes:

- `id` - table ID number
- `product_title` - title of laptop from web source
- `brand` - brand of laptop
- `model` - model of laptop
- `operating_system` - operating system pre-installed on laptop
- `extended_title` - an amalgamation of `product_title`, `brand`, `model`, and `operating_system`

Amazon Table: **3000 tuples**
Walmart Table: **4847 tuples**
Total number of tuple pairs (A X B): 14,541,000 pairs

Blocking Step

Blocker Debugging

The original blocker we used was the default overlap blocker. For this iteration, we blocked on the attribute `model` and used a q-gram tokenizer with q = 5 and an overlap size requirement of 8. This gave us a total of 2,597 tuple pairs, which was a little too restrictive. Using the `debug_blocker` functionality in Magellan, we observed that blocking on brand alone seemed incorrect.

We next tried blocking on `extended_title` since it contains information of many of the attributes, and again used the q-gram tokenizer with q = 5 but a much larger overlap size of 25. This gave us a a reduction on A X B on the order of 45 times smaller. However, on debugging the blocker again we still observed that it was dropping potential matches, so we changed our blocker again.

Final Blocker

Because the default blocker wasn't achieving the level of matching that we wanted, we developed our own custom (black box) blocker with matched on the `extended_title` attribute and computes a Jaccard score to measure potential matches. We used q-grams with q = 3 and a threshold of 0.5 when comparing matches. The full code can be found in our notebook (`em_notebook`). Debugging this final blocker seemed to not drop many potential matches and also reduced the candidate set by a lot, so we stopped here to proceed with the matching step. We sampled 450 tuple pairs for labeling (i.e. set G), wrote them to a CSV file, and labeled them there for use in the matching step. Additionally, we split this set into our train and test sets I and J (respectively).

Number of tuple pairs in candidate set after blocking: **7,326 tuple pairs**
Number of tuple pairs that were labeled from sample: **450 tuple pairs**

Classifier Metrics for Learning Methods

To begin training using our six learning methods provided in Magellan, we created a feature table F using the auto-generated features from Magellan and converted our training set I insto a set H of feature vectors. Running on each of the learning methods let to the following metrics:

Matcher	Average Precision	Average Recall	Average F-1
Decision Tree	0.931906	0.849995	0.888477
Random Forest	0.923791	0.940841	0.931533
Support Vector Machine	0.872856	0.970963	0.918780
Linear Regression	0.894443	0.935375	0.913853
Logistic Regression	0.888296	0.932618	0.908653
Naive Bayes	0.919928	0.773327	0.839341

Based on our selection criteria of a learning method with the highest Recall given that Precision that was at least 90%, we chose to select the **Random Forest** matcher.

Cross Validation Debugging

Debugging Iteration I

Matcher to debug:
Random Forest Precision: 92.38%, Recall: 94.08%, F-1: 93.15%

Making features from non-attributes, such as the IDs of each row, resulted in many false positive matches due to high similarity between IDs. Additionally, our tables had some dirty attributes with erroneous/inconsistent values resulting in several false negatives. We first dropped these problematic features from the auto-feature generation and tried different tokenizers for matching (3, 5, and 10). After reconverting I into a set H of feature vectors, we found the Random Forest matcher to have precision of 90.88%, recall of 92.91%, and F-1 of 91.80%. We then ran all matchers through cross validation and recomputed the metrics which are displayed in the table below.

Matcher	Average Precision	Average Recall	Average F-1
Decision Tree	0.910556	0.912291	0.911003
Random Forest	0.908812	0.929106	0.917972
Support Vector Machine	0.827146	0.988857	0.899620
Linear Regression	0.898446	0.973204	0.933452
Logistic Regression	0.884140	0.988857	0.932342
Naive Bayes	0.911270	0.927804	0.918882

Debugging Iteration II

Matcher to debug:
Random Forest Precision: 90.88%, Recall: 92.91%, F-1: 91.80%

The previous iteration seemed to have actually reduced the metrics for Random Forest but it still remained to be the best matcher. We deduced that a lot of useful information was actually being lost because of the data being so dirty. To combat this, we tried adding more features using custom functions. Specifically, we dropped `product_title`, `model`, and `operating_system` and wrote functions that would create value-features returning 1 if both OR neither tuples' attribute contained any of the passed in values and 0 otherwise. We used this to generate many more features and by hard-coding several common values for `brand`, `model`, and other indicative values, and passed this to our feature generating function. We extended it to use value sets and wrote another function that would return 1 if both tuples contained the same value for any of the values in the value set. We reconverted I into H (feature vectors) and found the Random Forest matcher to have much improved scores with precision of 96.36%, recall of 97.20%, and F-1 of 96.73%. We then ran all matchers through cross validation and recomputed the metrics which are displayed in the table below.

Matcher	Average Precision	Average Recall	Average F-1
Decision Tree	0.954048	0.937980	0.945281
Random Forest	0.963564	0.972000	0.967336
Support Vector Machine	0.893285	0.992000	0.939676
Linear Regression	0.962487	0.952694	0.957178
Logistic Regression	0.913782	0.960000	0.935088
Naive Bayes	0.946487	0.942230	0.943907

As shown by the scores above, the previous debugging iteration seemed to greatly improve the precision, recall and F-1 score, so we stopped further debugging.

Final Matcher Selected:
Random Forest Precision: 96.36%, Recall: 97.20%, F-1: 96.73%

Final Metrics

After training all of the matchers on I, we then computed all of the metrics for their performance on the test set J, as summarized by the following table.

Matcher	Precision	Recall	F-1
Decision Tree	93.62%	94.62%	94.12%
Random Forest	94.51%	92.47%	93.48%
Support Vector Machine	80.56%	93.55%	86.57%
Linear Regression	93.68%	95.70%	94.68%
Logistic Regression	88.78%	93.55%	91.1%
Naive Bayes	93.55%	93.55%	93.55%

Final Matcher Y on test set J:
Random Forest Precision: 94.51%, Recall: 92.47%, F-1: 93.48%

Time Estimates

Blocking

approximately **5 days** (debugging and reblocking)

Labeling

approximately **1 day** (labeling 450 tuple pairs)

Matcher Debugging

approximately **3 days** (debugging and creating new features to test on)

Recall Discussion

Our final recall on the test set J was 92.47%, which is reasonably high but could have been higher. To improve this metric, we could have been more liberal in our blocking step to allow more potential matches, but this would come at the cost of largely expanding the potential set of matches. Since our original tuple pair table was 14.5 million, we wanted to restrict the computation by blocking more heavily, but we could relax this constraint a bit to improve recall.

Magellan Comments, Extra Credit

1. All blockers have fixed thresholds, which doesn't make sense when an attribute can be extremely short or long. Instead, there should be an option for the threshold to be a fraction, or to use similarity measures. I know that you can use py_entitymatching, but you can't specify the q-value here.

2. There should also be an option for the comparisons to be case-insensitive, without having to change our own data to all be lower case.

3. extract_feature_vecs can put NaNs into the table. There was no message of how to fix it, even though the fix was simple, there was no way to know what the fix was right away.

4. There should be an option to combine candidate sets with intersection, not only union.

5. In the last line of the first example at there should be "atypes2" instead of "atype2" as listed.

6. There should be an explanation that says you should use “em.to_csv_metadata(A,filename)” instead of “A.to_csv(filename)”