Entity Matching Songs and Tracks with Magellan

Lokananda Dhage, Mary Feng, Varun Naik

CS 838 Project Stage 3

1. Entity Type to Match, the Tables, Number of Tuples per Table

   Since the dataset that we used for previous project stages had fewer than 3,000 tuples in each of the two tables, we switched to the movie-track-song dataset described on the project page. We performed the development stage of entity matching between the Song and Track tables, which we used “as is” from the project page. We considered a song tuple and a track tuple to be a match if they both described the same real-world song. The Song table contains 961,593 tuples, and the Track table contains 734,485 tuples. To make the sizes of the tables more manageable, we downsampled using the em.down_sample() with size 5,000 and y_param 1. The downsampled A and B are sample_A.csv and sample_B.csv, respectively. There are 4,192 tuples in sample_A and 5,000 tuples in sample_B. The columns of the downsampled A and B are the same as in the original dataset. Since the provided dataset did not define the notion of a match, we used our own definition. Two tuples are a match if their song titles were very similar and the artists overlapped. We noticed that often times, there was only one artist in songs.csv, but tracks.csv was more descriptive and often contained multiple artists. For example, if tuple A has title “Chinese Burn (Lunatic Calm mix)” and artist “Curve”, while tuple B has title “Chinese Burn (Lunatic Calm Mix)” and artist “toni halliday+dean garcia+curve”, this is considered a match since the song titles are very similar, and the artist “Curve” of tuple A is in the artist “toni halliday+dean garcia+curve” of tuple B. Similarly, if tuple A has title “A Swingin Safari” with artist “Bert Kaempfert And His Orchestra” and tuple B has title “A Swingin Safari (The MATCH GamE)” with artist “bert kaempfert”, this is considered a match since they refer to the same real world song. However, if two tuples shared song titles but artists were different (as in, no overlap), then it was not a match. For example, if tuple A has title “(When You Feel Like You're In Love) Don't Just Stand There” by artist “Carl Smith” and tuple B has the same title but artists “cherokee jack henley+ernest tubb+roy face+hal smith”, this is not considered a match because artist “Carl Smith” from tuple A is not in artists “cherokee jack henley+ernest tubb+roy face+hal smith” of tuple B. Finally, different versions of the same song by the overlapping artists are considered a match. Tuple titles may contain something like “Live” to denote it is the live version. If the titles are similar enough and artists overlap, then this is still considered a match. For example, if tuple A has title “Honey Don't (Live)” with artist “Carl Perkins”, and tuple B has title “Honey Dont” with artist “carl perkins”, this is still considered a match since this refers to the same song by the same artists.

2. Blocker Used and Number of Tuple Pairs in Candidate Set After Blocking (5223)

   In the blocking step, we identified all (song, track) pairs such that song.title shared at least one word with track.song, and song.artist_name shared at least one word with track.artists. We noticed that Magellan's overlap blocker only performs blocking on a single column, so for convenience, we initially performed black-box blocking. However, this approach was prohibitively slow, taking roughly 20 minutes to run on the downsampled tables. Clearly, black-box blocking would not scale to the entirety of the Song and Track tables during the production stage.

   Therefore, we performed overlap blocking on the Song and Track tables, with a minimum word overlap count of 1 between Song.title and Track.song. Next, we performed overlap blocking on the candidate set, with a minimum overlap count of 1 between Song.artist_name and Track.artists. Finally, we performed black box blocking to remove false matches, as the overlap blocker in Magellan incorrectly converted certain Unicode characters to ASCII (e.g. 'á' became 'a'). We made sure that this chain of blockers produced exactly the same output C as the naive black-box blocker. C contained 5,223 candidates.

3. Number of Tuple Pairs in Sample G that were Labeled

   As described in the Magellan how-to guide, we sampled and labeled 500 candidates from candidate set C to produce G.csv. Then, we split G.csv into a training set I.csv (350 candidates) and a test set J.csv (150 candidates).

4. Precision, Recall, F1 when Performing Cross Validation (for the first time)

   We used 10 fold cross validation.

   Precision:

	Name	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Fold 8	Fold 9	Fold 10	**Mean score**
0	DecisionTree	10	0.833333	1.0	1.000000	1.0	0.500000	1.000000	0.75	1.000000	1.0	0.6	**0.868333**
1	RF	10	1.000000	1.0	0.800000	1.0	0.571429	1.000000	0.75	1.000000	1.0	1.0	**0.912143**
2	SVM	10	1.000000	0.0	1.000000	1.0	0.000000	0.000000	0.50	1.000000	0.0	0.0	**0.450000**
3	LinReg	10	1.000000	1.0	0.833333	1.0	0.571429	1.000000	0.75				
4	LogReg	10	1.000000	1.0	1.000000	1.0	0.666667	0.666667	0.60				
5	NaiveBayes	10	0.750000	1.0	0.833333	1.0	0.500000	0.500000	0.75				

3	1.000000	1.0	1.0	**0.915476**
4	1.000000	1.0	1.0	**0.893333**
5	0.857143	0.8	1.0	**0.799048**

Recall:

	Name	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Fold 8	Fold 9	Fold 10	**Mean score**
0	DecisionTree	10	0.8	1.00	0.25	1.000000	1.0	1.0	1.0	0.666667	1.0	0.8	**0.851667**
1	RF	10	0.8	1.00	1.00	1.000000	1.0	1.0	1.0	0.666667	1.0	0.6	**0.906667**
2	SVM	10	0.0	0.25	0.00	0.000000	0.2	0.2	0.0	0.333333	0.0	0.2	**0.118333**
3	LinReg	10	0.4	1.00	1.00	0.833333	0.6	1.0	1.0	0.666667	1.0	0.6	**0.810000**
4	LogReg	10	0.6	1.00	1.00	1.000000	1.0	1.0	1.0	0.666667	1.0	0.8	**0.906667**
5	NaiveBayes	10	0.6	1.00	1.00	1.000000	0.8	1.0	1.0	0.666667	1.0	1.0	**0.906667**

F1:

	Name	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Fold 8	Fold 9	Fold 10	**Mean score**
0	DecisionTree	10	0.888889	0.833333	0.923077	0.666667	0.857143	0.923077					
1	RF	10	0.750000	0.909091	0.833333	0.800000	0.923077	0.923077					
2	SVM	10	0.400000	0.000000	0.000000	0.000000	0.000000	0.250000					
3	LinReg	10	0.750000	0.909091	0.923077	0.800000	0.600000	1.000000					
4	LogReg	10	0.888889	0.769231	0.923077	0.800000	0.833333	0.923077					
5	NaiveBayes	10	0.888889	0.833333	0.923077	0.500000	0.923077	1.000000					

0	0.857143	0.800000	0.923077	1.0	**0.867241**
1	0.750000	1.000000	0.923077	1.0	**0.881166**
2	0.500000	0.500000	0.250000	0.0	**0.190000**
3	0.571429	1.000000	0.923077	1.0	**0.847667**
4	0.666667	1.000000	1.000000	1.0	**0.880427**
5	0.444444	0.857143	0.857143	1.0	**0.822711**

5. Learning Based Matcher Selected After Cross Validation
   RF had the second highest precision after LinReg (**0.912143** vs 0.915476), but its recall was tied for highest at **0.906667** (along with LogReg and NaiveBayes, and its F1 was highest at **0.881166**, so we chose RF.

6. Debugging Iterations and Cross Validation Iterations Performed
   a. Debugging RF matcher; P: 0.912143, R: 0.906667, F1: 0.881166
   b. We used the RF GUI debugger and found that there were errors in labeling. To rectify this, we looked over the labeling to check for errors and fix them. In the dev set I, there were 5 errors found; all were instances where the tuple pairs were a match (should be 1) but were marked as not a match (mistakenly labeled as 0). These were lines 69, 113, 212, 303, and 328 in the dev set I. The errors were fixed by simply changing to label for 5 lines to 1. This fixed dev set is I.csv.
   c. Precision/Recall/F1 after fixing labeling errors:
      Precision:

	Name	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7
0	DecisionTree	10	0.857143	1.0	0.800000	1.0	1.000000	1.00	0.833333
1	RF	10	1.000000	1.0	0.833333	1.0	0.857143	1.00	1.000000
2	SVM	10	1.000000	0.0	1.000000	1.0	0.000000	0.00	1.000000
3	LinReg	10	1.000000	1.0	0.833333	1.0	0.857143	1.00	1.000000
4	LogReg	10	1.000000	1.0	1.000000	1.0	1.000000	1.00	0.800000
5	NaiveBayes	10	0.750000	1.0	0.833333	1.0	0.750000	0.75	1.000000

	Fold 8	Fold 9	Fold 10	**Mean score**
0	1.000000	0.833333	1.0	**0.932381**
1	1.000000	1.000000	1.0	**0.969048**
2	1.000000	0.000000	0.0	**0.500000**
3	1.000000	1.000000	1.0	**0.969048**
4	1.000000	0.666667	1.0	**0.946667**
5	0.857143	0.833333	1.0	**0.877381**

Recall:

	Name	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Fold 8	Fold 9	Fold 10	**Mean score**
0	DecisionTree	10	0.666667	0.833333	1.0	0.833333	1.000000	1.0	1.0	0.75	1.0	0.8	**0.888333**
1	RF	10	0.666667	0.833333	1.0	1.000000	1.000000	1.0	1.0	0.75	1.0	0.8	**0.905000**
2	SVM	10	0.000000	0.166667	0.0	0.000000	0.333333	0.2	0.5	0.25	0.0	0.2	**0.165000**
3	LinReg	10	0.500000	0.833333	1.0	1.000000	0.666667	1.0	1.0	0.75	1.0	0.6	**0.835000**
4	LogReg	10	0.666667	0.833333	1.0	1.000000	1.000000	1.0	1.0	0.75	1.0	0.8	**0.905000**
5	NaiveBayes	10	0.666667	0.833333	1.0	1.000000	0.833333	1.0	1.0	0.75	1.0	1.0	**0.908333**

F1:

	Name	Num folds	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Fold 8	Fold 9	Fold 10	**Mean score**
0	DecisionTree	10	0.888889	0.857143	0.923077	0.666667	1.000000	0.923077	0.800000	0.857143	1.000000	0.0	**0.791600**
1	RF	10	0.888889	0.833333	0.923077	0.750000	0.923077	1.000000	0.888889	1.000000	0.923077	1.0	**0.913034**
2	SVM	10	0.333333	0.000000	0.444444	0.400000	0.000000	0.250000	0.400000	0.500000	0.250000	0.0	**0.257778**
3	LinReg	10	0.888889	0.833333	0.923077	0.857143	0.727273	1.000000					
4	LogReg	10	1.000000	1.000000	0.923077	0.857143	0.833333	0.923077					
5	NaiveBayes	10	1.000000	0.923077	0.923077	0.600000	0.923077	1.000000					

3	0.750000	1.000000	0.923077	1.0	**0.890279**
4	0.888889	0.750000	0.923077	1.0	**0.909860**
5	0.600000	0.857143	0.857143	1.0	**0.868352**

7. Final Best Matcher Selected, Its Precision/Recall/F1
   RF was tied with LinReg for highest precision, as both were **0.969048**. RF had second highest recall after NaiveBayes, but the difference was quite small (**0.905000** versus 0.908333). Finally, RF had the highest F1 score at **0.913034**. Since RF performed well in all these metrics, it was chosen to be the final best matcher selected.

8. Precision, Recall, F1 on Test Set J
   a. For each of the six learning methods, precision/recall/F1 on test set J of training the matcher based on that method on I
      DT:
      Precision : 92.31% (12/13)
      Recall : 85.71% (12/14)
      F1 : 88.89%
      RF:
      Precision : 92.31% (12/13)
      Recall : 85.71% (12/14)
      F1 : 88.89%
      SVM:
      Precision : 100.0% (2/2)
      Recall : 14.29% (2/14)
      F1 : 25.0%
      LinReg:
      Precision : 92.31% (12/13)
      Recall : 85.71% (12/14)
      F1 : 88.89%
      LogReg:
      Precision : 85.71% (12/14)
      Recall : 85.71% (12/14)
      F1 : 85.71%
      NaiveBayes:
      Precision : 80.0% (12/15)
      Recall : 85.71% (12/14)
      F1 : 82.76%
   b. Precision/recall/F1 on test set J of final best matcher Y selected (RF)
      RF:
      Precision : 92.31% (12/13)
      Recall : 85.71% (12/14)
      F1 : 88.89%

9. Approximate Time Estimates

   All of the following runtime measurements are on a quad-core Macbook Pro with a 2.7 GHz processor and 16 GB of RAM.

   a. To do the blocking

      Writing the overlap blocker took roughly 6 hours. It ran on the downsampled tables in about 14 seconds.

   b. To label the data

      Labeling the sample of 500 took approximately 25 minutes.

   c. To find the best matcher

      It took approximately 3 hours to decide on a matcher, debug it, and complete this process. Runtime to do 10 fold cross validation and testing on test set J for all 6 ML matchers is approximately 12 seconds.

10. Why Recall is Not Higher and Steps to Obtain Higher Recall

    Looking at some of the False negatives gave us an intuition why the recall was not higher. We identified the following two reasons for False Negatives.

    1. Generally the song title of one of the tables of a candidate pair has additional information, although a portion of the string matches exactly with the song title in the other table, resulting in low overlap and jaccard scores.

    2. Presence of large number of special characters in the song title of one of the table in the candidate pair.

    Steps to obtain higher recall

    1. We observe that the automatically generated features such as jaccard score, edit distance used to match the title of the songs were not enough to identify a matches as described in reason 1 above.

       Hence we could add a new feature which checks for the length of the overlapping substring and have a high threshold on this. This will ensure that the False negatives arising out of the reason 1 can be eliminated at the same time having a high threshold can prevent the possibility of False positives. Such a threshold can be learnt a priori or a Domain expert could determine it.

    2. We observe that the extra information in one of the Song/Track titles is generally is enclosed in parenthesis “()”. For eg. "Love Is All Around (The Mary Tyler Moor Show Theme)" vs "Love Is All Around". Such strings could be preprocessed to remove the extra information before sim measures are calculated between the pair of Song titles. This can reduce the false negatives arising due to reason 1.

    3. To overcome the False Negatives due to reason 2 we could perform Data cleaning on the song titles to remove spurious/special characters present in the song titles.

11. Bonus: Comments on Magellan

- The how-to guide did a great job of explaining how the different parts of Magellan (downsampling, blocking, labeling, matching, etc.) fit together. It might be useful to have a single motivating example throughout the guide, to show what the development stage might look like in real life. The document would be more clear if there were fewer “letters” - for example, U represents a matcher, the user, and a training set in different parts of the guide. Lastly, there were minor typos and grammar errors throughout.

- Magellan itself would benefit from different verbosity levels for the various commands. For example, while performing blocking, it would have been useful to see information about the candidates as they were generated, in addition to a progress indicator. As mentioned in the documentation, it would be very useful to have more ways to combine blockers, in addition to union and chaining. Furthermore, for the overlap blocker, we needed to adjust the stopwords and punctuation characters manually; it would have been better if they were a parameter in the constructor.

- The GUI Labeler used during matching two tuples could be designed in a more user friendly way, by aligning the attribute values of the tuple one below the other or by placing corresponding attributes from the two tables side by side.