Project Stage III

Entity matching in structured data

Team

Anna Chang
Sripradha Karkala
Simmi Pateriya

Data Description

We used the songs and tracks tables from the movie-track-song dataset that was provided with this project stage. Each tuple in these tables describe a song entity. The original tables contained 961,593 tuples and 734,485 tuples for songs and tracks respectively. We reduced the size to 17,731 (sample_A) and 10,000 (sample_B) after downsampling.

Definition of Match

For this project, we considered a song and track to match if the following conditions were fulfilled:

1. The **artist_name** in the Songs table matched at least one of the **artists** in the Track table.
2. The **title** attribute in Songs clearly referred to the same **song** in Tracks.

Remixes of multiple songs were not considered matches and were removed from the golden data. Covers were kept in the golden data but were only considered matches if the **artist_name** and **artists** had at least one match.

Blockers

We initially created two blockers:

1. 1-word overlap blocker on **artist_name** and **artists** (variable C1 in Jupyter)
2. 1-word overlap blocker on **title** and **song** (variable C2 in Jupyter)

The results of these blockers were combined in variable C and the debugging output was saved as dbg.csv. None of the 200 tuples generated by the debugger appeared to be matches, so we sampled 450 tuples from C and saved them to S. We found too many non-matches in this sample taken for labelling. Since, there were over 1 million tuple pairs in C, we tried applying a rule based blocking on C to get better sample for labelling as mentioned below.

We tried a 3-gram Jaccard on **title** and **song** (blocker D). The debugger (dbg2) showed that this rule was removing too many matches so we decided to go back to the overlap blocker and see if we could apply domain knowledge to improving it.

There were some foreign songs, so we began by adding more stop words ('de', 'del', 'du', 'of', 'la', 'le') to the overlap blocker E. The output showed that many songs also share commons words including pronouns as well as words like ‘theme’ and ‘love’, so we expanded the stop word list to include the following: ['i', 'im', 'my', 'me', 'you', 'your', 'we', 'our', 'el', 'after', 'theme', 'y', 'that', 'like', 'little', 'all', 'love'].

This blocker (E1) had a fairly decent debugger output (dbg3.csv). However there were some tuples that were clearly matches but were being block such as:

ltable_title	ltable_artist_name	rtable_song	rtable_artists
Between Raising Hell and Amazing Grace	Big & Rich	Between Raising Hell and Amazing Grace	big+rich

We surmised that big+rich was being tokenized as “bigrich” and therefore wasn’t matching “Big & Rich”, so we modified sample_B by replacing all “+” in the **artists** column with a single white space. This new table was called sB1.

We did another 1-word overlap on **artist_name** and **artists** (C1a), this time using sA and sB1. By our definition of a match, both **artists** and **title** needed to have at least one overlap, so we did an additional candidate set 1-word overlap block on **title** and **song** (stored as C1b). The debugger (dbg4) showed good results, but the candidate set (C1b) was still too large.

We did further candidate blocking on C1b by writing a function that blocks tuples whose first two words of the title don’t have at least one word in common. For example, “Feed me now” and “Now ain’t that spiffy” would be blocked but “Hungry like a wolf” and “I’m hungry” would be candidate tuple pairs. This blocker was stored as C3. The debugger (dbg5) indicated that some good matches were still being blocked. These were tuple pairs whose titles had some variation such as:

ltable_title	ltable_artist_name	rtable_song	rtable_artists
The Monkees (Theme From)	The Monkees	(Theme From) The Monkees	tommy boyce bobby hart the monkees
Instrumental Intro - Ferry Cross the Mersey (Live)	Gerry And The Pacemakers	Ferry Cross the Mersey	gerry marsden gerry the pacemakers

To keep these tuple pairs, we combined C3 together with a new blocker, C4, to create our final blocker (C5) that left candidate pairs with the following characteristics:

1. There is a 1-word overlap on the artist and 1-word overlap on the title
2. Either one of the first two words of song match or the song titles have at least 2 words in common

The debugger (dbg6) still had 2 matches out of 200, but we were willing to sacrifice these matches for a more manageable **candidate set of 2742 tuple pairs**.

Sample G

We sampled 400 tuple pairs from C5 and labelled 390 of them. 10 were discarded due to ambiguity. The final set contained 184 positive examples and 206 negative examples.

Performance metrics on I - First iteration cross validation

Learning Method	Precision	Recall	F-1
Decision Tree	0.873993	0.890479	0.876245
Random Forest	0.939561	0.907212	0.919469
SVM	0.985714	0.532327	0.675387
Linear Regression	0.927188	0.93514	0.929112
Logistic Regression	0.9262	0.937981	0.929215
Naive Bayes	0.924872	0.884185	0.899696

Matcher Selected

After cross validation on I, we selected linear and logistic regression as the best matchers.

Debugging process

We tried to debug the linear and logistic regression models but it appeared that they were unsupported and the following AttributeError appeared:

```
# Debug linear regression
lr = em.LogisticRegressionMatcher()
lr.fit(table=H1,
        exclude_attrs=['_id', 'ltable_id', 'rtable_id', 'gold'],
        target_attr='gold')

pred_table = lr.predict(table=H1,  exclude_attrs=['_id', 'ltable_id', 'rtable_id', 'gold'],  append=True, target_attr='
eval_summary = em.eval_matches(pred_table, 'gold', 'predicted_labels')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-301-5cb3780cc0e9> in <module>()
      1 # Debug linear regression
----> 2 lr = em.LogisticRegressionMatcher()
      3 lr.fit(table=H1,
      4         exclude_attrs=['_id', 'ltable_id', 'rtable_id', 'gold'],
      5         target_attr='gold')

AttributeError: 'module' object has no attribute 'LogisticRegressionMatcher'
```

Instead, we debugged by looking at what feature vectors were being considered. Originally, we had these correspondences:

[('id', 'id'), ('title','song'), ('artist_name','artists'), ('year', 'year')]

But since ID seemed to be arbitrarily assigned, we decided to remove any features involving ID and reduce the correspondences to:

[('title','song'), ('artist_name','artists'), ('year', 'year')]

We reran cross-validation for all models on H1 (no id features) and saw an improvement in precision/recall/f1 for both linear and logistic regression. Linear regression slightly outperformed logistic regression, so we chose it as our final matcher.

Learning Method	Precision	Recall	F-1
Decision Tree	0.882197	0.908698	0.891535
Random Forest	0.946532	0.905813	0.923309
SVM	0.984615	0.467998	0.61687
Linear Regression	0.94185	0.942832	0.940952
Logistic Regression	0.933343	0.945673	0.936345
Naive Bayes	0.931641	0.884185	0.903597

Performance metrics: Train on I, Test on J

After training on the entire Set I, we tested on J. The decision tree and linear regression models performed the best:

Learning Method	Precision	Recall	F-1
Decision Tree	0.9286	0.8814	0.9043
Random Forest	0.8824	0.8824	0.8182
SVM	1	0.4237	0.5952
Linear Regression	0.9444	0.8644	0.9027
Logistic Regression	0.9273	0.8644	0.8947
Naive Bayes	0.8909	0.8909	0.8596

Performance metrics: J (Final)

There didn't seem to be a built in function for debugging linear regression, so we debugged on the decision tree. Based on the types of false positives we saw, we decided not to try to increase the precision further.

One false positive was due to incorrect labelling and half of the false positives (4 out of 8) were from the classical music genre. The challenge with these tuples is that many songs share multiple common words, but a single word can make a large difference. For example, Sonata in D Minor is a different piece than Sonata in C Minor. Other genres generally have more distinct titles so it would be difficult to adjust for classical pieces without potentially reducing precision for other types of music. This could be fixed if the raw data included a genre attribute.

Since we made no adjustments to our matcher, our best matchers were still the decision tree and linear regression:

Learning Method	Precision	Recall	F-1
Decision Tree	0.9286	0.8814	0.9043
Linear Regression	0.9444	0.8644	0.9027

Time estimates

We spent a significant amount of time setting up, reading documentation, and debugging both the blockers and matchers.

Task	Time (hours)
Blocking	12
Labelling	2
Matching	5

Recall

Based on the false negatives we saw when debugging the decision tree, 4/10 tuple pairs had parenthesis in the name of the song for one tuple but not for the other. Below are a few examples:

ltable_title	ltable_artist_name	rtable_song	rtable_artists
A Garden In The Rain (Digitally Remastered)	The Four Aces	A Garden in the Rain	al alberts the four aces carroll gibbons james dyrenforth
God Only Knows (Sax Solo)	The Beach Boys	God only knows	brian wilson tony asher the beach boys

We tried to account for these during the blocking stage by adding ‘version’ to the stop words, but a better way would have been to add a feature to the matcher that does overlap, jaccard, and other string comparisons with song names that have the parenthesis and all the words in between them stripped.
Another idea for improving recall would be to treat the **artists** in the Tracks table as a list rather than a string and to add a feature that checks if the **artist_name** is an exact match with at least one of the **artists**:

ltable_title	ltable_artist_name	rtable_song	rtable_artists
What Do You Want From Me	Forever The Sickest Kids	WHAT DO YOU WANT FROM ME	caleb turman austin bello jonathan cook kent garrison forever the sickest kids

Lastly one quick method for improving recall would be to check for exact matches on both **artist_name/artists** and **title/song**. This could be done as a postprocessing step rather than adding it to the matcher. This would fix 2/10 false negatives:

ltable_title	ltable_artist_name	rtable_song	rtable_artists
The Passenger	Iggy Pop	The Passenger	iggy pop
IPT2	Battles	IPT2	battles

Feedback on Magellan

The Strengths
Magellan covers the whole entity matching process from manipulating the source data to analyzing matchers, and the debugger tools help users quickly find where mistakes occur for the entire entity matching process (i.e. both blocking and matching). Magellan makes the process fast because the user can quickly try various options available for debugging and matching, and this takes far less time than it would have taken if done manually. Another strength is that the main data structure used is the pandas dataframe which is well documented and familiar for most users.

Installation Suggestions
We faced some glitches in the installation process. Two of our team members found changes made to the filesystem on their machines during the installation process. It would be good if more detailed and updated instructions were made available for windows installation.

Documentation Suggestions
For the most part, the documentation was very helpful. There were certain functions that were used in the Jupyter notebook guides but seem to be missing from the API. For example, the guide demonstrated how to add stopwords to the blockers, but there weren't more details in the documentation. One hiccup we faced was that originally some of the words we added to the stopword list were capitalized and it took us some time to realize the overlap blocker will convert a string to lowercase before comparing it to the the stopwords. In other words, if we had "I" as a stopword, it wouldn't catch the I in something like "I left my heart in San Francisco." Instead, we had to add it to the stopword list as a lowercase "i".

Features/Capabilities Suggestions
It would be great if Magellan could give recommendations at each step to the user such as which blocker or matcher it thinks would be best to use. Based on previous tests, it might be able to make such suggestions by looking at the schemas for the tables, missing values etc.

Additionally, there were some matchers that we couldn't debug, such as linear and logistic regression. It would nice if users can manipulate LogisticRegressionMatcher() and LinearRegressionMatcher() the same way they can DTMatcher(). Currently we get an AttributeError if we try to call these functions although they're referenced under the "Specifying ML-Matchers and Performing Matching" section.

Another feature that would be appreciated is being able to pass in a tokenizer into the blockers. As mentioned in the blocking section of our writeup, the overlap blocker implementation seemed to be removing '+' characters and joining the remaining characters. It would be helpful the tokenization rules could be easily customized and if the default tokenization process were more transparent.

Magellan is effective because it builds on existing libraries (pandas, scikit-learn, etc). The challenge is when errors occur when py_entitymatching calls some of these external libraries and the error messages lack details or when the fixes to two errors seem incompatible. For instance, the overlap blocker produced an error message indicating that some strings couldn't be parsed because they weren't in 'utf-8' format. We went back and added this encoding attribute to the tables when they were being loaded, but then the rule-based blocker had trouble with some strings since they weren't compatible with the ascii codec. We also got a KeyError when debugging the blocker, but loading the candidate set from CSV seemed to have fixed the problem although the root of this problem is still a bit of a mystery.

Final Thoughts
There's a slight learning curve with using Magellan, but the numerous guides are very helpful and overall this toolkit really streamlines the entity-matching process.