CS 839 Data Science

Project Stage 3

＜Group 14＞

1. The names of all team members.
   - Wen-Fu Lee (wlee256@wisc.edu)
   - Yuan-Ting Hsieh (yhsieh28@wisc.edu)
   - Yahn-Chung Chen (chen666@wisc.edu)

2. Describe the type of entity you want to match, briefly describe the two tables (e.g., where did you obtain these tables), list the number of tuples per table.
   - The type of entity: movie
     - The type of entity: movie
     - The two tables
       - One is for IMDB website, and the other one is for Rotten Tomatoes website
       - In these two tables, both include the following attributes
         - movie_no: movie number
         - movie_name: movie name
         - movie_year: movie year
         - movie_certificate: movie certificate
         - movie_runtime: movie runtime
         - movie_genre: movie genre
         - movie_score: imdb score
         - movie_gross: movie gross
         - movie_director: movie directors
         - movie_star: movie stars
         - movie_writer: movie writer
         - tomatoter: tomatoter review socre
         - audience: audience review score
     - The number of tuples
       - IMDB: 3000
       - Rotten Tomatoes: 3012

3. Describe the blocker that you use and list the number of tuple pairs in the candidate set obtained after the blocking step.
   - First step, use overlap block by movie name, require at least one token is overlap, after this step we get around 778769 tuple pairs
   - Second step, block by movie star name, need to have at least two tokens of name overlap. For example, “David Lee, Marc Donald” and “David Lee, Gigi” would remain because “David” and “Lee” counts as two token. After this step we get 20356 tuple pairs

- Third step, block by the offset of movie release year, so that means their year's difference should be within a range to be considered as a potential match pair, so if offset is 2, then (2015, 2017) would remain while (2000, 2006) would be blocked. After this step we get 5833 tuple pairs
- Fourth step, block by cosine similarity of movie name, the cosine similarity of the pair's movie name should exceed a threshold. We set the threshold as 0.4. After this step we get 1938 tuple pairs and complete our total blocking procedure

4. List the number of tuple pairs in the sample G that you have labeled.
    - 510

5. For each of the six learning methods provided in Magellan (Decision Tree, Random Forest, SVM, Naive Bayes, Logistic Regression, Linear Regression), report the precision, recall, and F-1 that you obtain when you perform cross validation for the first time for these methods on I.

k = 5				
	Matcher	**Average precision**	**Average recall**	**Average f1**
0	DT	0.984615	1.000000	0.992157
1	RF	0.984615	0.992857	0.988520
2	SVM	0.991667	0.896529	0.941052
3	LinReg	0.984615	0.977473	0.980520
4	LogReg	0.984615	0.992857	0.988520
5	NB	0.984615	0.985165	0.984599

6. Report which learning based matcher you selected after that cross validation.
    - Decision Tree (with the highest F1 value)

7. Report all debugging iterations and cross validation iterations that you performed. For each debugging iteration, report (a) what is the matcher that you are trying to debug, and its precision/recall/F-1, (b) what kind of problems you found, and what you did to fix them, (c) the final precision/recall/F-1 that you reached. For each cross validation iteration, report (a) what matchers were you trying to evaluate using the cross validation, and (b) precision/recall/F-1 of those.
    - We applied 5-fold cross validation and fortunately got good precision and recall in our first try. Since most of our matchers reached the goal of 90% precision, we didn't do any debugging iteration. Instead, we focused on dealing with the falsely positive data, and the only problem we found among our data is labeling error.

- After fixing error, we tried different cross validation among all the matchers. The number of fold we tried are range from 3 - 7. We finally select Decision Tree as our matcher because it reached the highest F-1 value while k = 5.

k = 3				
	Matcher	**Average precision**	**Average recall**	**Average f1**
0	DT	0.976332	0.968992	0.971661
1	RF	0.984091	0.992424	0.988205
2	SVM	0.990741	0.896749	0.941329
3	LinReg	0.991667	0.969521	0.980094
4	LogReg	0.984259	0.984496	0.984099
5	NB	0.991667	0.984672	0.988028

k = 4				
	Matcher	**Average precision**	**Average recall**	**Average f1**
0	DT	0.984375	0.991379	0.987678
1	RF	0.984375	0.992857	0.988440
2	SVM	0.991379	0.895145	0.940221
3	LinReg	0.984375	0.966995	0.974804
4	LogReg	0.984375	0.992857	0.988440
5	NB	0.984375	0.984236	0.984054

k = 5				
	Matcher	**Average precision**	**Average recall**	**Average f1**
0	DT	0.984615	1.000000	0.992157
1	RF	0.984615	0.992857	0.988520
2	SVM	0.991667	0.896529	0.941052
3	LinReg	0.984615	0.977473	0.980520

4	LogReg	0.984615	0.992857	0.988520
5	NB	0.984615	0.985165	0.984599

k = 6				
	Matcher	**Average precision**	**Average recall**	**Average f1**
0	DT	0.986389	0.983333	0.984506
1	RF	0.986389	0.967754	0.975835
2	SVM	0.992424	0.894626	0.940670
3	LinReg	0.986389	0.967754	0.975835
4	LogReg	0.986389	0.984420	0.985075
5	NB	0.986389	0.984420	0.985075

k = 7				
	Matcher	**Average precision**	**Average recall**	**Average f1**
0	DT	0.984962	0.993506	0.988956
1	RF	0.984962	0.982540	0.983270
2	SVM	0.992063	0.894016	0.939571
3	LinReg	0.984962	0.973016	0.977992
4	LogReg	0.984962	0.992063	0.988196
5	NB	0.984962	0.982540	0.983270

8. Report the final best matcher that you selected, and its precision/recall/F-1.
   - Decision Tree (k = 5)
   - Precision: 0.984615
   - Recall: 1.000000
   - F1: 0.992157

9. Now report these numbers:
   For each of the six learning methods, train the matcher based on that method on I, then report its precision/recall/F-1 on J.

	Matcher	Average precision	Average recall	Average f1
0	DT	98.18%	96.43%	97.3%
1	RF	96.36%	94.64%	95.5%
2	SVM	98.0%	87.5%	92.45%
3	LinReg	98.18%	96.43%	97.3%
4	LogReg	96.49%	98.21%	97.35%
5	NB	96.49%	98.21%	97.35%

For the final best matcher Y selected, train it on I, then report its precision/recall/F-1 on J.

- Decision Tree
- Precision: 98.18%
- Recall: 96.43%
- F1: 97.3%

10. Report approximate time estimates: (a) to do the blocking, (b) to label the data, (c) to find the best matcher.
    - (a): The hours we spent: 10 hours (including normalizing), runtime: < 2 mins
    - (b): The hours we spent: 1.5 hours, runtime: 0 mins
    - (c): The hours we spent: 10 hours, runtime: < 1 mins

11. Provide a discussion on why you didn't reach higher recall, and what you can do in the future to obtain higher recall.
    - For this stage, our final recall achieves 96.43% on J, which is pretty high as a result. The possible reason is that the data sources, IMDB and Rotten Tomatoes websites, provide very clean and accurate information about movies. That is, there is no misled or erroneous information in the movie descriptions. In addition, movies' attributes are very discriminative. For example, we almost can judge if the two movies are the same according to their movie names and release years. After all, it's kind of impossible to have two movies with the same name released in the same year. We think that, in the future, we may try other data sources, which provide less information or are less correct, to make this learning task more realistic. After all, in the real life, we cannot assume that we can always get so ideal data sources.

12. BONUS POINTS: provide comments on what is good with Magellan and what is bad, that is, as users, what else would you like to see in Magellan. Are there any features/capabilities that you would really like to see being added? Any bugs? Depending on how detailed and helpful these comments are, you can get bonus point from 1-10 (which will help with the final grade, not just with the project).
    - The overall using result is good. The tutorials/guides in github repository is really helpful. The blocking and matching is easily done using Magellan. One suggestion

is that these ipynb tutorials are not on the project front page of Magellan, which will lower its chance to be spotted. Maybe you can consider putting those ipynbs' link at the front page of Magellan.

- Additional point to documentation is that, maybe in the one page doc, you could also add link refer to the corresponding ipython-notebook guide, because we think sometimes the word documentation is not precise/detail enough
- One thing we found is a bit inconvenient and confusing is that if we want to normalize the DataFrame by some pandas function after we read the data by Magellan's reading function, it will cause problem in Magellan's config/metadata. But this is not intuitive, because we will think that after we drop certain rows that have missing values, it should not affect the functionality of entity matching. To reproduce the bug, you can use Magellan to read in csv file first, and then use pandas function to drop some rows, and then when you go to blocking phase, there will be problems occured. This thing actually cause us a lot of time.
- A suggestion to the above problem is either fix that bug or the documentation should point out explicitly that you should not drop rows after you use Magellan read or it will cause problem