CS839 - Project Stage 3 Report
Blocking rules and estimating precision and recall

Team Members:

- Sri Harshal Parimi (sparimi@wisc.edu)
- Shebin Roy Yesudhas (royyesudhas@wisc.edu)
- Sankarshan Umesh Bhat (sbhat6@wisc.edu)

Entity of choice:
The entity we choose to extract was movie information of different genres from the IMDB website and Rotten Tomatoes website.

Table A: IMDB(imdb.csv): The table from IMDB contains movies information (Name, ReleaseDate, Certificate, Rating, Runtime, Director, Genre, Grossing).

Table B: Rotten Tomatoes (rotton.csv): The table from Rotten Tomatoes contains movies information (Name, ReleaseDate, Certificate, Rating, Runtime, Director, Genre, Grossing).

Tuples:
IMDB: 3000
Rotten Tomatoes: 3072

Schema:

- ❏ Name: The name of the movie.
- ❏ ReleaseDate: Data of release of the movie.
- ❏ Certificate: Content rating provided for the movie
- ❏ Rating: The average rating based on fan reviews for the movie
- ❏ RunTime: Running length in minutes of the movie
- ❏ Director: The director(s) of the movie
- ❏ Genre: Conventional category which identifies the movie

Blocking rules:
The candidate set size downloaded from CloudMatcher turned out to be 72165 tuple pairs. We labeled 50 tuple pairs from the candidate set and got the density to be 4/50 = **0.08.**

We analyzed the candidate set and realized that matching tuple pairs in our dataset almost always agreed on similar/same **movie name and director.** However, attributes like '**Genre'** took a set of values that were not lexically similar between the two tables(Ex- Mystery, adventure vs Thriller). Attributes like '**Rating'** had different scales between both the tables(Ex- IMDB rated movies on a scale of 10 and Rotten Tomatoes on a scale of 5). Time-related attributes like **'RunTime'** and **'ReleaseDate'** had minor inconsistencies between the tables and there were a lot of missing values for old movies. So it made a lot of sense to just use **Name** and **Director** attributes for employing blocking rules in the first iteration.

We attempted the **Overlap Blocker** implementation available in **py_entitymatching** to drop tuple pairs. Tuple pairs that had atleast **1 word-level overlapping token** in both **'Name'** and **'Director**' attributes were retained but the rest were dropped by the blocker. This simple rule was good enough to reduce the candidate set size to **2123** tuple pairs.

After this step, we ran the **debug blocker** module of the provided **Jupyter notebook** link to check for **true positives that were being dropped** by this newly incorporated blocking rule. We noticed only a few such cases(Ex- Tuple pairs for the 'Iron Man 2' movie had different values for the 'Director' attribute and hence they were dropped by the blocking rule).

Next, we used the reduced candidate set to sample another 50 tuple pairs. We got the new density to be 23/50 = **0.46.** This is greater than 0.2 and hence we stopped our blocking process at this juncture.

Estimating precision and recall:
We used the provided Jupyter notebook to compute precision and recall from the following data:

1) Reduced candidate set of 2123 tuple pairs obtained by incorporating blocking rules.
2) 400 Sampled tuple pairs(100 sampled for blocking + 300 additionally sampled from the reduced candidate set).
3) Prediction list downloaded from CloudMatcher.

We obtained the following results:
Recall = [0.9371096866388409 - 0.9910340259360095]
Precision = [0.9186143717366582 - 0.9782780006778256]