CS 839 Data Science

Project Stage 4

＜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. How did you combine the two tables A and B to obtain E? Did you add any other table? When you did the combination, did you run into any issues? Discuss the combination process in detail, e.g., when you merge tuples, what are the merging functions (such as to merge two age values, always select the age value from the tuple from Table A, unless this value is missing in which case we select the value from the tuple in Table B).
   - First, we applied the pipeline from stage 3 over all our data and we got a set called T with around 740 matching tuples inside. The problem we have here is there are some duplicative matching in T. (e.g. A1 matches B2 and also matches B3).
   - To deal with this problem, we removed all the tuples with duplicate matching and put them into a set called D. After that, there are 713 1-on-1 matching tuples left in T. We then used the “_id” attribute which is the ID number of each pair in T as the key to merge data from table A and table B. Since A and B have no "_id" attribute, we first left joined A and B with T by "ltable_movie_no" and "rtable_movie_no" respectively. It can add the "_id" attribute to all the elements in A and B.
   - With the "id", we can then left join A by B on the "_id" to merge tables. We select attributes "movie_name, movie_year, movie_certificate, movie_runtime, movie_genre, movie_score, movie_gross, movie_director, movie_star" from A and attributes "movie_writer, tomatoter, audience" from B. We called the merged table T’.
   - We also removed movie listed in T’ from A, B to generate A’, B’ which have no matching pair between them. For that reason, elements in A’ and B’ have some missing values.
   - Because the confidence score for each of duplicative matching tuples in D are all 1.0, it seems no better way to handle data in D than manually checking or randomly picking. Here we randomly picked tuples and used hash table to track movie number for preventing duplicate. After this process, we got a new set with only 1-on-1 matching tuples inside. We applied the same merging function mentioned above to merge this set and got the merged table D'.
   - Because we randomly picked tuples from previous stage, there are some elements could not be paired after the process. We called these remaining elements as R_A (form table A) and R_B (from table B).
   - Finally, we combined A’, R_A, B’, R_B, T’, D’ to form the Table E.

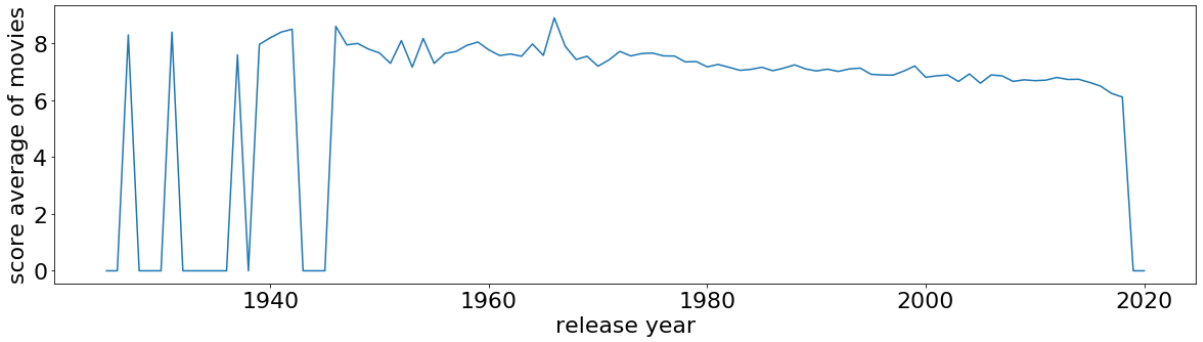
3. Statistics on Table E: specifically, what is the schema of Table E, how many tuples are in Table E? Give at least four sample tuples from Table E.
    - Schema:
        - movie_name
        - movie_year
        - movie_certificate
        - movie_runtime
        - movie_genre
        - movie_score
        - movie_gross
        - movie_director
        - movie_star
        - movie_writer
        - tomatoter
        - audience
    - 5277 rows, 724 matching tuples in Table E
    - Sample:

Schema \ Index	0	2506	5064	5180
movie_name	ready play one	a touch of sin	sing street	short term 12
movie_year	2018	2015	2016	2013
movie_certificate	PG-13	NR	PG-13	R
movie_runtime	140min	125min	106min	96min
movie_genre	Action, Adventure, Sci-Fi	Drama	Comedy, Drama, Music	Drama
movie_score	7.9	NaN	8	8
movie_gross	$68.35M	NaN	$3.23M	$1.10M
movie_director	Steven Spielberg	Zhangke Jia	John Carney	Destin Daniel Cretton
movie_star	Tye Sheridan, Olivia Cooke, Ben Mendelsohn, Lena Waithe	Tao Zhao, Wang Baoqiang, Jiang Wu, Lanshan Luo, Zhang Jiayi, Li Meng	Ferdia Walsh-Peelo, Aidan Gillen, Maria Doyle Kennedy, Jack Reynor	Brie Larson, Frantz Turner, John Gallagher Jr., Kaitlyn Dever
movie_writer	NaN	Zhangke Jia	John Carney	Destin Daniel Cretton

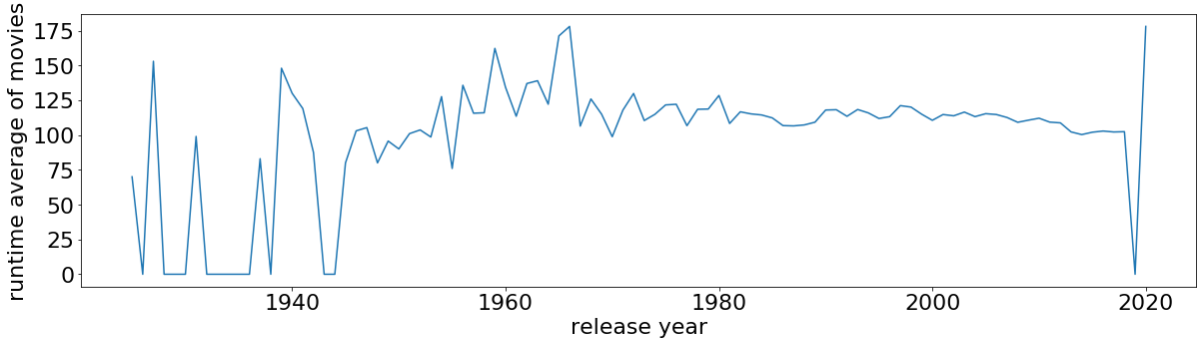
tomatoter	NaN	93%	95%	99%
audience	NaN	74%	92%	93%

4. Append the code of the Python script (that merges the tables) to the end of this pdf file.
    - Refer to the last page of this document.

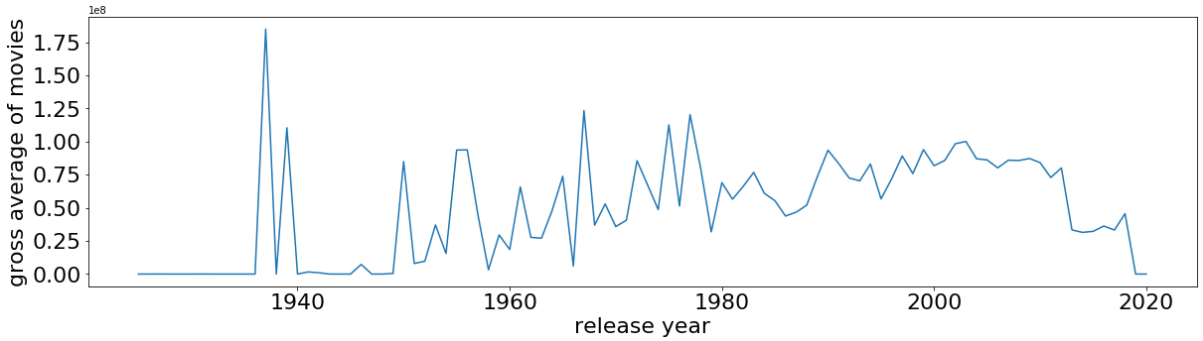
5. What was the data analysis task that you wanted to do? (Example: we wanted to know if we can use the rest of the attributes to accurately predict the value of the attribute loan_repaid.) For that task, describe in detail the data analysis process that you went through.
    - Task
        - We want to know if we can use 3 attributes, movie_score, movie_runtime, movie_gross, to accurately predict if a movie was released after 1980s.
    - First, why we decided to adopt these 3 features and this target is based on the observation below. For the first diagram of score average vs release year, we can see that there is a decreasing score average after 1980s.



    - For the second diagram of runtime average vs release year, we can see that the runtime average after 1980s is a little shorter than those before 1980s.



    - For the third diagram of gross average vs release year, we can see that the gross average after 1980s is much higher than those before 1980s.



- Note that when calculating the average data above, we skipped those movies with missing values in these 3 attributes in order to have a more unbiased view.
- Then, we can take this problem as a classification task
  - Features
    - movie_score, movie_runtime, movie_gross
  - Label
    - 1 means a movie was released after 1980s
    - 0 means if a movie was released before 1980s
  - Training setting
    - Classifier candidates: Decision Tree, Random Forest, SVC
    - Cross validation: 5 folds
    - Choosing best classifier based on F1
    - Sample number: 1817
  - Testing setting
    - Using best classifier for prediction
    - Sample number: 605

6. Give any accuracy numbers that you have obtained (such as precision and recall for your classification scheme).
   - Training:

	Decision Tree	Random Forest	SVC
Precision	0.950456	0.950903	0.942700
Recal	0.974314	0.983654	0.998832
F1	0.962237	0.967001	0.969955

   - Testing:

	SVC
Precision	0.953488
Recal	0.996528
F1	0.974533

7. What did you learn/conclude from your data analysis? Were there any problems with the analysis process and with the data?
    - Based on the accuracy shown above, we can conclude that we actually can adopt only 3 attributes, movie_score, movie_runtime, movie_gross, to accurately predict if a movie was released after 1980s. We think that one problem with this task is how to handle the missing values in these 3 values. The current accuracy is obtained by removing those movies with the missing values. If we still want to keep those movies for the classification training, we may need to consider more aspects about how to fill in those missing values, which is somehow hard to decide.

8. If you have more time, what would you propose you can do next?
    - If we have more time, we want to propose how to fill in missing values in each attribute of movies since in a real life it is a common issue: data could be missing somewhere, no matter it's due to human errors or data corruption. With this mechanism of filling in missing values, we believe that our learning model could give a more robust performance for prediction in any situations.

Code:

```
import pandas as pd
import os
import py_entitymatching as em
import numpy as np

datasets_dir = os.getcwd() + os.sep

pathA = datasets_dir + "/data/imdb_clean.csv"
pathB = datasets_dir + "/data/tomato_clean.csv"
pathC = datasets_dir + "/data/block.csv"

A = pd.read_csv(pathA)
B = pd.read_csv(pathB)
# Rename first empty attr
# df.rename(columns={"Unnamed: 0": "id"},  inplace=True)

p_A = A[['movie_no', 'movie_name', 'movie_year', 'movie_director', 'movie_star']]
p_B = B[['movie_no', 'movie_name', 'movie_year', 'movie_director', 'movie_star']]

em.set_key(p_A, 'movie_no')
em.set_key(p_B, 'movie_no')

pathS = datasets_dir + "/data/labeled_data.csv"

S = em.read_csv_metadata(pathS,
                         key='_id',
                         ltable=p_A, rtable=p_B,
                         fk_ltable='ltable_movie_no', fk_rtable='rtable_movie_no')

IJ = em.split_train_test(S, train_proportion=0.7, random_state=0)
I = IJ['train']
J = IJ['test']

# Classifier
dt = em.DTMatcher(name='DecisionTree', random_state=0)
svm = em.SVMMatcher(name='SVM', random_state=0)
rf = em.RFMatcher(name='RF', random_state=0)
lg = em.LogRegMatcher(name='LogReg', random_state=0)
ln = em.LinRegMatcher(name='LinReg')
nb = em.NBMatcher(name='NB')
```

```
# Feature generation
F = em.get_features_for_matching(p_A, p_B, validate_inferred_attr_types=False)

H = em.extract_feature_vecs(I,
                            feature_table=F,
                            attrs_after='label',
                            show_progress=False)

# Missing value
H = em.impute_table(H,
                exclude_attrs=['_id', 'ltable_movie_no', 'rtable_movie_no', 'label'],
                strategy='mean')

# Corss Validation
result = em.select_matcher([dt, rf, svm, ln, lg, nb], table=H,
        exclude_attrs=['_id', 'ltable_movie_no', 'rtable_movie_no', 'label'],
        k=5, # Num of fold
        target_attr='label', metric_to_select_matcher='f1', random_state=0)

# Apply to testing set

fc = result['selected_matcher'] # LinReg here
# dt = em.DTMatcher(name='DT', random_state=0)

fc.fit(table=H,
       exclude_attrs=['_id', 'ltable_movie_no', 'rtable_movie_no', 'label'],
       target_attr='label')

L = em.extract_feature_vecs(J, feature_table=F,
                            attrs_after='label', show_progress=False)

predictions = fc.predict(table=L, exclude_attrs=['_id', 'ltable_movie_no', 'rtable_movie_no',
'label'],
                append=True, target_attr='predicted', inplace=False)

eval_result = em.eval_matches(predictions, 'label', 'predicted')
em.print_eval_summary(eval_result)


# C is the blokced data
C = em.read_csv_metadata(pathC,
                         key='_id',
                         ltable=p_A, rtable=p_B,
                         fk_ltable='ltable_movie_no', fk_rtable='rtable_movie_no')

cl = result['selected_matcher']
```

```
L = em.extract_feature_vecs(C, feature_table=F,
                            show_progress=False)

predictions = cl.predict(table=L, exclude_attrs=['_id', 'ltable_movie_no', 'rtable_movie_no'],
              append=True, target_attr='predicted', probs_attr='score', return_probs = True,
inplace=False)

# Tuple predicted as match
tn = predictions[(predictions['predicted'] == 1)]

# Duplicate detect
dup_A = tn[tn["ltable_movie_no"].duplicated(keep=False)]
dup_B = tn[tn["rtable_movie_no"].duplicated(keep=False)]

dup_T = pd.merge(dup_A, dup_B, how='outer', on=['_id','ltable_movie_no', 'rtable_movie_no'])

# Remove duplicate
# clean_dup_A = tn["ltable_movie_no"].drop_duplicates(keep='first')
# clean_dup_B = tn["rtable_movie_no"].drop_duplicates(keep='first')
clean_dup_A = tn[~(tn["ltable_movie_no"].isin(dup_T['ltable_movie_no']))]
clean_dup_B = tn[~(tn["rtable_movie_no"].isin(dup_T['rtable_movie_no']))]
# clean_dup_B

# Remove all matched from A and B (A' and B')
new_A = A[(~A['movie_no'].isin(clean_dup_A['ltable_movie_no']))]
new_B = B[(~B['movie_no'].isin(clean_dup_B['rtable_movie_no']))]

# Select one pair of duplicate pairs
dict_A = {}
dict_B = {}

res = []
s_A = []
s_B = []
for index, row in dup_T.iterrows():
    if (row['ltable_movie_no'] not in dict_A) and (row['rtable_movie_no'] not in dict_B):
        dict_A[row['ltable_movie_no']] = 1
        dict_B[row['rtable_movie_no']] = 1
        s_A.append(row['ltable_movie_no'])
        s_B.append(row['rtable_movie_no'])
        res.append(row['_id'])

dict_RA = {}
```

```
dict_RB = {}
for index, row in dup_T.iterrows():
    if (row['ltable_movie_no'] not in dict_A):
        dict_RA[row['ltable_movie_no']] = 1;
    if (row['rtable_movie_no'] not in dict_B):
        dict_RB[row['rtable_movie_no']] = 1;


dict_RB
rem_B = B[B['movie_no'].isin(dict_RB)]

dict_RA
rem_A = A[A['movie_no'].isin(dict_RA)]


# s_A = clean_dup_A['ltable_movie_no'].append(pd.Series(s_A))
# s_B = clean_dup_B['rtable_movie_no'].append(pd.Series(s_B))
# rem_B
m_A = A[(A['movie_no'].isin(clean_dup_A['ltable_movie_no']))]
m_B = B[(B['movie_no'].isin(clean_dup_B['rtable_movie_no']))]
clean_dup_A.rename(columns={"ltable_movie_no": "movie_no"}, inplace = True)
clean_dup_B.rename(columns={"rtable_movie_no": "movie_no"}, inplace = True)

m_B = pd.merge(m_B, clean_dup_B[['_id', 'movie_no']], how = 'left', on = 'movie_no')
m_A = pd.merge(m_A, clean_dup_A[['_id', 'movie_no']], how = 'left', on = 'movie_no')
m_F = pd.merge(m_A[['_id','movie_no', 'movie_name', 'movie_year', 'movie_certificate',
'movie_runtime',
                    'movie_genre', 'movie_score', 'movie_gross', 'movie_director', 'movie_star']],
               m_B[['_id','movie_writer', 'tomatoter', 'audience']], how = 'left', on = '_id')
# m_F
m_Dl = tn[tn["_id"].isin(res)]
m_Dr = m_Dl.copy()
m_Dl.rename(columns={"ltable_movie_no": "movie_no"}, inplace = True)
m_Dr.rename(columns={"rtable_movie_no": "movie_no"}, inplace = True)
m_DA = A[(A['movie_no'].isin(s_A))]
m_DB = B[(B['movie_no'].isin(s_B))]
m_DB = pd.merge(m_DB, m_Dr[['_id', 'movie_no']], how = 'left', on = 'movie_no')
m_DA = pd.merge(m_DA, m_Dl[['_id', 'movie_no']], how = 'left', on = 'movie_no')
m_DF = pd.merge(m_DA[['_id','movie_no', 'movie_name', 'movie_year', 'movie_certificate',
'movie_runtime',
                    'movie_genre', 'movie_score', 'movie_gross', 'movie_director', 'movie_star']],
               m_DB[['_id','movie_writer', 'tomatoter', 'audience']], how = 'left', on = '_id')
new_A = new_A.drop(['Unnamed: 0', 'movie_no'], axis = 1)
new_B = new_B.drop(['Unnamed: 0', 'movie_no'], axis = 1)
rem_A = rem_A.drop(['Unnamed: 0', 'movie_no'], axis = 1)
```

```
rem_B = rem_B.drop(['Unnamed: 0', 'movie_no'], axis = 1)
m_F = m_F.drop(['_id', 'movie_no'], axis=1)
m_DF = m_DF.drop(['_id', 'movie_no'], axis=1)
frames = [new_A, rem_A, new_B, rem_B, m_F, m_DF]

result = pd.concat(frames, ignore_index=True)

# result

result.to_csv("merge_table.csv", encoding='utf-8')
```