CS839 Project Stage 4: Integration and Analysis

Date: 05/09/2018 Team name: Big HIT

Team members: **H**oai Nguyen, **I**saac Sung, **T**rang Vu

Summary: This report describes how we have integrated two tables of movie data and performed analysis on them. We matched entities between the two tables by using the entity matching process we completed in Project Stage 3. Table 1 below summarizes the minimal information necessary for this report, and the sections proceeding that explain our project in further detail.

Table 1: Project summary

Entity type	Movies from internet movie databases	
Table E	**Attribute Name**	**Merging Rule**
	title	Keep title from A. If title from B is not an exact match from A, append B's title to the alternative title field if B's title is not already in A's alternative title field.
	directors	Append directors from B to A such that the directors in merged directors are unique. For example, if directors from A is (a; b), from B is (a; c), the directors in merged table should be a;b;c
	writers	Same rule as directors
	cast	Same rule as directors
	genres	Same rule as directors
	keywords	Same rule as directors
	content_rating	Keep content_rating from A. If A is missing value here, use B's.
	run_time	Take average and round to the nearest integer
	release_year	Keep release_year from A (should be matched exactly due to blocking rule anyway)
	languages	Same rule as directors
	rating	Take average after converting the rating in B to a scale of 10
	budget	Take the average and round it to the nearest integer
	revenue	Take the average and round it to the nearest integer
	opening_weekend_revenue	Take the average and round it to the nearest integer
	production_companies	Same rule as directors
	production_countries	Same rule as directors

	alternative_titles	Same rule as directors + rule for title
Table E Statistics	**Schema**	Given above.
	Number of Tuples	1,111 tuples

	Sample Tuples	`(0,The Shape of Water,Michael Shannon;Sally Hawkins;Michael Stuhlbarg;Octavia Spencer;Richard Jenkins,Guillermo del Toro,Vanessa Taylor;Guillermo del Toro,Thriller;Horror;Fantasy;Adventure;Romance;Drama,interspecies romance;underwater scene;creature;mute woman;sign language;capture;1960s;bathtub;government;orphan;fairy tale;cold war;baltimore;laboratory;fishman;scientist,R,123,2017,Russian;English;French;American Sign Language,7.5,$19450000,$159291046,$166564,Bull Productions;Double Dare You (DDY);Fox Searchlight Pictures,USA,La forma del agua)`
		
		`(1,"Three Billboards Outside Ebbing, Missouri",Kerry Condon;Caleb Landry Jones;Frances McDormand;Alejandro Barrios;Sam Rockwell;John Hawkes;Peter Dinklage;Woody Harrelson,Martin McDonagh,Martin McDonagh,Crime;Drama,police brutality;anger;small town;black comedy;billboard;missouri;murder;police corruption;police violence;teenage girl;rape;guilty conscience,R,115,2017,English,8.2,$15000000,$155999956,$322168,Blueprint Pictures;Film 4;Fox Searchlight Pictures,UK;USA,3 Billboards)`

Data Analysis Task	For the data analysis task, we decided to perform an OLAP approach to discover any correlations between movies' budget and revenue.	
Results	There is a positive correlation between movies' budget and revenue (Pearson's correlation coefficient of 0.815).	
Conclusion	If a movie has a large budget, there is a high chance that it will generate a lot of revenue.	

1. Table Merging

We merged Tables A and B using a variety of rules for each attribute. Many of them used a rule of basically taking the union of all the values, since many of the attributes contained multiple people's names (e.g. cast would have a set of actors).

Merging Rules

This set of rules describe how to merge the tuples from tables A and B:

General rule: If an attribute in A is missing a value, the merged value takes value from B and vice versa. If the attribute's values are empty in both tuples, the merged value is also empty.

1) title: keep title from A. If title from B is different from A, append B's title to the alternative title field if B's title is not already in A's alternative title (see rule for alternative_title for more detail)
2) directors: append directors from B to A such that the directors in merged directors are unique. For example, if directors from A is (a; b), from B is (a; c), the directors in merged table should be a;b;c. When comparing pair of strings (ie. director names), use Smith Waterman similarity score normalized to the smaller string length threshold at 0.6.

3) writers: same rule as directors
4) cast: same rule as directors
5) genres: same rule as directors
6) keywords: same rule as directors
7) content_rating: keep value from A
8) run_time: take average and round it to nearest integer
9) release_year: keep value from A (value B in B should be exact match since release_year was used as attribute-equivalent blocking criteria)
10) languages: same rule as directors
11) rating: divide value in B by 10 and take the average
12) budget: If value in A has different currency symbol from value in B, keep the value with US dollar symbol ($). If they have the same symbol, take the average and round to nearest integer.
13) revenue: same rule as budget
14) opening_weekend_revenue: same rule as budget
15) production_companies: same rule as directors
16) production_countries: same rule as directors
17) alternative_titles: if title from B is different from title and alternative titles from A, (based on Smith Waterman similarity score normalized by smaller string length threshold at 0.8), append B's title to alternative_titles.

Issues during merging tables:

1) Mismatches existed in the predicted matches produced by the classifier. Therefore, we had to resolve the mismatches by manually removing all of the incorrectly matched tuple pairs.
2) We also had to refine merging rules when combining multiple-value attributes (eg. directors, cast, etc.) such that partial matching was allowed. This was necessary in order to avoid adding unnecessary instances in merged directors in cases with obvious encoding issues such as 'Alejandro G. IÃ±Ã¡rritu; Alejandro GonzÃ¡lez IÃ±Ã¡rritu'.
3) Mismatches in currency symbols existed in 'budget' and 'revenue'. In order to merge these values correctly we had to check for the matching currency symbol before taking an average.

2. Table E Statistics

The schema for Table E is as follows:

1. title: the title of the movie
2. directors: a list of all directors credited to this movie
3. writers: a list of all writers credited to this movie
4. cast: a list of the top-billed cast of actors for this movie
5. genres: a list of the genres (action, comedy, horror, etc.) describing this movie
6. keywords: a list of thematic keywords to describe this movie
7. content_rating: the MPAA rating for the movie (e.g. G, PG, PG-13)
8. run_time: the running time of the movie in minutes
9. release_year: the year the movie was released
10. languages: the languages associated with the movie
11. rating: the user rating of the movie
12. budget: the estimated budget value of the movie
13. revenue: the estimated total revenue of the movie
14. opening_weekend_revenue: the estimated revenue for the movie in its opening weekend
15. production_companies: a list of the production companies associated with the movie

16. production_countries: a list of the countries associated in the production of the movie
17. alternative_titles: a list of all the alternative titles that are commonly associated with this movie

3. Data Analysis Task

We were interested in identifying and conducting correlation discovery between movies' budget and revenue amounts.

Process

We first profiled the budget and revenue data to see how many missing values there were. We then computed the mean, standard deviation, minimum, and maximum values to get an idea of what the data was like. The standard deviation in the budget and revenue amounts were significantly large. This was to be expected since many independent films have tiny budgets compared to Hollywood blockbusters that have budgets in the range of hundreds of millions of dollars.

A major challenge we faced in our analysis of budget and revenue data were when values contained different currency symbols than the US dollar symbol. In order to process foreign currencies, there are a variety of difficulties that make the process non-trivial. We considered doing a simple currency exchange using the current conversion rate; however, due to inflation and some foreign currencies becoming obsolete over the years (e.g. the Russian Ruble RUR or the Austrian Schilling), the currency exchange may skew or misrepresent the data. Also, if we considered inflation for foreign countries' data, then we would also have to convert the dollar rate for older US movies as well. In order to make this process easier, we decided to only look at movies that had budget and revenue in US dollars. Also, to reduce variation in data due to inflation, we restricted the timeframe to movies released within the last 20 years.

We summarized the data by taking the average value of budget and revenue for each year, and plotted a bar graph (see Figure 1). By looking at the bar graph, we were not able to see any particular trends because of the large amount of variation.

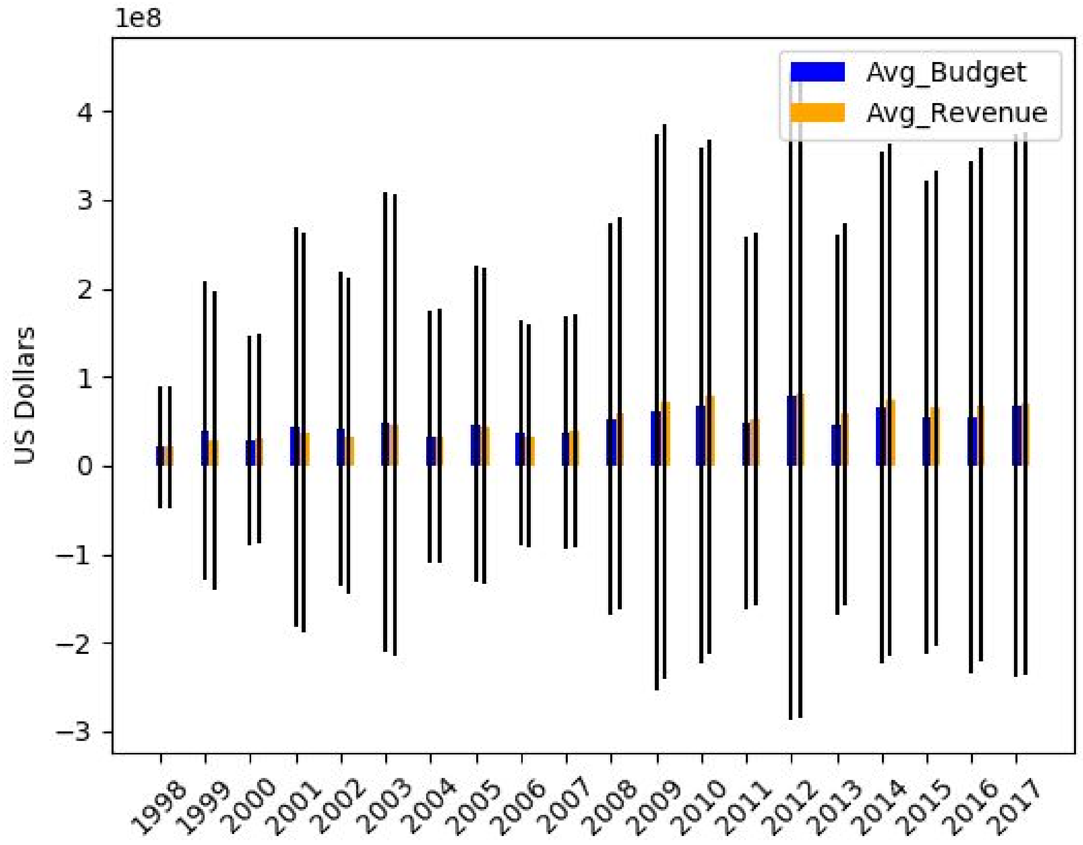

Figure 1. Bar graph of average budget and average revenue split by year.
There are huge variances, and so no meaningful relationships can be seen in the data in this graph.

We then plotted all of the data within the last 10 years on a scatter plot, colored by year (see Figure 2). We noticed that there is some sort of linear correlation existing between budget and revenue. To confirm our intuition, we computed Pearson's correlation coefficient for our data, which was 0.815. This indicated a positive linear correlation between budget and revenue and confirmed what we had seen with our eyes.

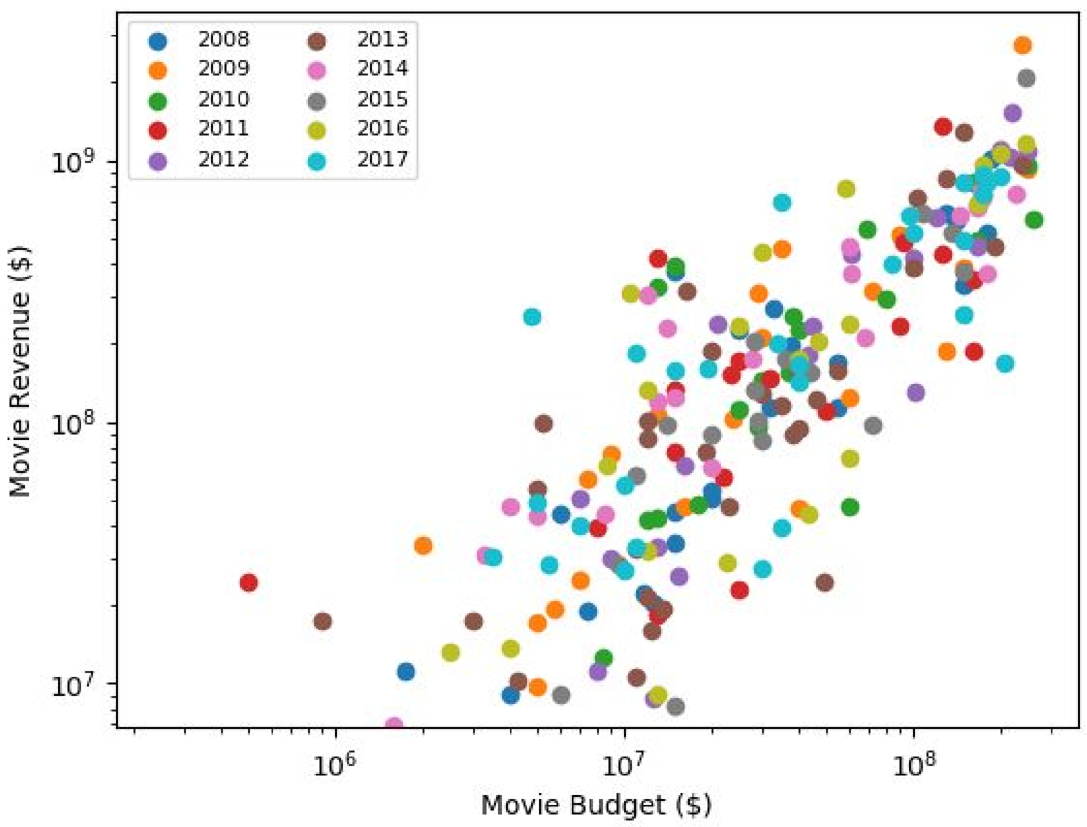

Figure 2. Scatter plot of revenue vs budget broken down by years.
A clear, linear correlation can be seen in the data.

Then, we decided to drill-down and zoom in on the year 2009 to identify the movie with the highest revenue and budget for that year. The movie in 2009 with the highest budget was Avatar directed by James Cameron for over $200 million, and it also had the highest revenue for that year as well at over $2.7 billion.

4. Results

This supports our conclusion that even though there is a lot of variation in budget and revenue for movies, based on our analysis, there is a linear correlation between budget and revenue. Spend more on a movie, and it is more likely that the movie will be a blockbuster with a large revenue. However, correlation does not imply causation, so we cannot say that higher budgets cause higher revenues.

5. Discussion

If we had more time, we would have built a ML classifier to see if we could accurately predict revenue based on budget and other features of a movie. It also would have been interesting to analyze the correlation between other features, such as opening_weekend_revenue. We could have also drilled-down and added more features to the analysis, such as which directors have the most positive correlation between budget and revenue.

References:

[1] IMDb: http://www.imdb.com/
[2] TMDb: https://www.themoviedb.org/?language=en

Appendix

A. Python Code to Merge Tables

```
# Import py_entitymatching package
import py_entitymatching as em
import os
import pandas as pd
import py_stringmatching as sm
import numpy as np

# Specify directory
FOLDER = './Data/'

# Set seed value (to get reproducible results)
seed = 0

def read_files():
    # Read in data files
    A = em.read_csv_metadata(FOLDER+'A.csv', key = 'id') # imdb data
    B = em.read_csv_metadata(FOLDER+'B.csv', key = 'id') # tmdb data
    C = em.read_csv_metadata(FOLDER+'C.csv', key = '_id', ltable = A, rtable = B,
                             fk_ltable = 'l_id', fk_rtable = 'r_id') # candidates that survive
blocking step
    G = em.read_csv_metadata(FOLDER+'G.csv', key = '_id', ltable = A, rtable = B,
                             fk_ltable = 'l_id', fk_rtable = 'r_id') # labeled data

    return A, B, C, G

def predict_matching_tuples(A, B, C, G):
    # Split G into I and J for CV
    IJ = em.split_train_test(G, train_proportion = 0.5, random_state = 0)
    I = IJ['train']
    # Generate features set F
    F = em.get_features_for_matching(A, B, validate_inferred_attr_types = False)
    # Convert G to a set of feature vectors using F
    H = em.extract_feature_vecs(I, feature_table = F, attrs_after = 'label',
                                show_progress = False)
    excluded_attributes = ['_id', 'l_id', 'r_id', 'label']
    # Fill in missing values with column's average
    H = em.impute_table(H, exclude_attrs = excluded_attributes,
                        strategy='mean')
    # Create and train a logistic regression - the best matcher from stage3.
    lg = em.LogRegMatcher(name='LogReg', random_state=0)
    lg.fit(table = H, exclude_attrs = excluded_attributes, target_attr = 'label')
    # Convert C into a set of features using F
    L = em.extract_feature_vecs(C, feature_table = F, show_progress = False)
    # Fill in missing values with column's average
    L = em.impute_table(L, exclude_attrs=['_id', 'l_id', 'r_id'],
                        strategy='mean')
    # Predict on L with trained matcher
    predictions = lg.predict(table = L,
                             exclude_attrs=['_id', 'l_id', 'r_id'],
                             append = True, target_attr = 'predicted',
                             inplace = False, return_probs = False,
                             probs_attr = 'proba')
    # Extract the matched pairs' ids
    matched_pairs = predictions[predictions.predicted==1]
    matched_ids = matched_pairs[['l_id', 'r_id']]
    # Save matched_pairs to file so we don't have to train and predict each time the code is
executed
    matched_ids.to_csv(FOLDER + 'predictedMatchedIDs.csv', index = False)

def resolve_mismatches(df):
    # remove manually inspected mismatches
    # Inspect 'l_id' and 'r_id' for potential mismatches.
    l_dup = df[df.duplicated(subset = 'l_id', keep = False)]
    r_dup = df[df.duplicated(subset = 'r_id', keep = False)]
    # Uncomment out the print statements below to view duplicates
    #print(l_dup)
    #print(r_dup)
    # After manually inspecting the duplicates, found the following mismatches:
    mismatches = [['a872', 'b2879'], ['a987', 'b3508'], ['a2730', 'b4185'], ['a1723', 'b386'],
['a1843', 'b223']]
    # Make a copy of matched_ids dataframe
    matchedIDs = df.copy()
    # Find indices of mismatched rows
    mismatched_indices = []
    for index, row in df.iterrows():
        for mismatch in mismatches:
            if row['l_id'] == mismatch[0] and row['r_id'] == mismatch[1]:
                mismatched_indices.append(index)
    #print(mismatched_indices)
    # Drop the mismatched rows from the matched table
    matchedIDs.drop(matchedIDs.index[mismatched_indices], inplace = True)

    return matchedIDs

def merge_tables(A, B, matchedIDs):
    # merge tables A and B into table E using matched IDs
    # first create a new data frame E with schema similar to A and B
    merged_data = []
    idx = 0
    for index, row in matchedIDs.iterrows():
        tupleA = A[A['id'] == row['l_id']]
        tupleB = B[B['id'] == row['r_id']]
        tupleE = merge_tuples(tupleA, tupleB, idx)
        merged_data.append(tupleE)
        idx = idx + 1
    E = pd.DataFrame(data = merged_data, columns = list(A))
    E.to_csv(FOLDER+'E.csv', index = False)

def merge_tuples(tupleA, tupleB, index):
    # merging two tuples A, B into tuple E
    attributes = list(tupleA)
    revenue_budget_attributes = ['budget', 'revenue', 'opening_weekend_revenue']
    multiple_value_attributes = ['directors', 'writers', 'cast', 'genres', 'keywords',
                                 'languages', 'production_companies','production_countries']
    merged_values = []
    for attribute in attributes:
        valueA = tupleA[attribute].values[0]
        valueB = tupleB[attribute].values[0]
        if attribute == 'id':
            merged_value = index
        elif attribute == 'title':
            merged_value = valueA
        elif attribute == 'content_rating':
            if pd.isnull(valueA):
                merged_value = valueB
            else:
                merged_value = valueA
        elif attribute == 'run_time':
            if pd.isnull(valueA):
                merged_value = valueB
            elif pd.isnull(valueB):
                merged_value = valueA
            else:
                merged_value = int(round((float(valueA) + float(valueB))/2))
        elif attribute == 'rating':
            if pd.isnull(valueA):
                merged_value = float(valueB)/10
            elif pd.isnull(valueB):
                merged_value = valueA
            else:
                merged_value = round((float(valueA) + float(valueB)/10)/2,1)
        elif attribute == 'release_year':
            merged_value = valueA
        elif attribute in revenue_budget_attributes:
            merged_value = merge_money(valueA, valueB)
        elif attribute in multiple_value_attributes:
            merged_value = merge_multiple_values_attribute(valueA, valueB)
        else:
            merged_value = merge_alternative_title(tupleA, tupleB)
        merged_values.append(merged_value)

    return merged_values

def merge_money(a, b):
    if pd.isnull(a):
        if not pd.isnull(b):
            symbolb, numberb = extract_symbol(b)
            if symbolb == '-':
                symbolb = ''
            return symbolb + numberb
        else:
            return a
    elif pd.isnull(b):
        if not pd.isnull(a):
            symbola, numbera = extract_symbol(a)
            if symbola == '-':
                symbolb = ''
            return symbola + numbera
        else:
            return a
    else:
        symbola, numbera = extract_symbol(a)
        symbolb, numberb = extract_symbol(b)
        if symbola == '-':
            symbola = ''
        if symbolb == '-':
            symbolb = ''
        if symbola != symbolb:
            if symbola == '$':
                return symbola + numbera
            elif symbolb == '$':
                return symbolb + numberb
            else:
                return symbola + numbera
        else:
            merged_value = int(round((float(numbera) + float(numberb))/2))
            return symbola + str(merged_value)

def extract_symbol(x):
    # extract the currency symbol from string x
    symbol = ''
    number = ''
    for i in range(0, len(x)):
        if not x[i].isdigit():
            symbol = symbol + x[i]
        else:
            number = number + x[i:]
            break
    return symbol, number

def merge_multiple_values_attribute(a, b):
    """Returned the merged content of the two cell a and b.
    a and be must be strings likes directors, writers, etc.
    We assume ';' as delimeters.
    Allow partial matching for string names using Jaccard score threshold at 0.8"""
    if (pd.isnull(a)):
        return b
    elif (pd.isnull(b)):
        return a
    else:
        a_list = list(set(a.split(';')))
        b_list = list(set(b.split(';')))
        merged_value = []
        for string_a in a_list:
            for string_b in b_list:
                score = compute_simScore(string_a, string_b)
                if score >= 0.6:
                    b_list.remove(string_b)
                else:
                    continue
        merged_value.extend(a_list)
        merged_value.extend(b_list)
        merged_value = ';'.join(merged_value)

        return merged_value

def compute_simScore(str1, str2):
    # compute similarity score between str1 and str2 using Smith Waterman measure
    sw = sm.SmithWaterman()
    return sw.get_raw_score(str1, str2)/min(len(str1), len(str2))

def merge_alternative_title(tupleA, tupleB):
    # merge alternative title
    altTitles = tupleA['alternative_titles'].values[0]
    altTitles = altTitles.split(';')
    titleA = tupleA['title'].values[0]
    titleB = tupleB['title'].values[0]
    simScore = compute_simScore(titleA, titleB)
    match = False
    # if titleB does not match titleA, considering add it to altTitles
    if simScore < 0.8:
        for title in altTitles:
            score = compute_simScore(title, titleB)
            if score >= 0.8:
                match = True
                break
        if match == False:
            # add titleB to altTitles if it does not match any of the alternative titles
            altTitles.append(titleB)

    merged_value = ';'.join(altTitles)

    return merged_value

def main():
    # read files
    A, B, C, G = read_files()
    # predict matching tuples if matchedIDs.csv is not in the Data folder.
    if not os.path.exists(FOLDER + 'predictedMatchedIDs.csv'):
        predict_matching_tuples(A, B, C, G)
    predicted_matchedIDs = pd.read_csv(FOLDER + 'predictedMatchedIDs.csv')
    # resolve any mismatches in the matched table
    matchedIDs = resolve_mismatches(predicted_matchedIDs)
    matchedIDs.to_csv(FOLDER+'ABmatches.csv', index = False)
    merge_tables(A, B, matchedIDs)

if __name__ == '__main__':
    main()
```