Stage 4 Report: Integrating and performing analysis

I. How we combined tables A and B to obtain E

After obtaining the set of matches between Tables A and B (refer to *matches.csv*), we defined several merging rules to merge tuples properly.

The following attributes in our dataset were the following: *Album*, *Artist*, *Genre*, *Label*, *Producer*, *Release Date*, and *Meta Score.* In turn, we defined the following rules/functions to merge tuples:

1. For attribute *Album* or *Artist*: Always select the longer string value to be the album name (or artist name).
2. For attributes *Genre, Label,* or *Producer*: Cell values in either one of the three attributes are represented as lists. So for each attribute, we merged by taking the union of the two lists. For lists that were empty, we replaced an empty list ('[]') with *NaN*.
3. For attribute *Release Date*: Always take the release date from the Wikipedia dataset.
4. For attribute *Meta Score*: Always take the metacritic score from the Metacritic dataset because the Wikipedia dataset does not provide this information.

We used the rules above to add the matches to Table E. After, we added all non-matches from the Metacritic dataset and Wikipedia dataset to Table E. Note that no other tables were added to Table E.

II. Statistics on Table E

Schema: E(Album, Artist, Genre, Label, Producer, Release Date, Meta Score).

Tuples: 5194 tuples.

Table 1: Sample tuples from Table E

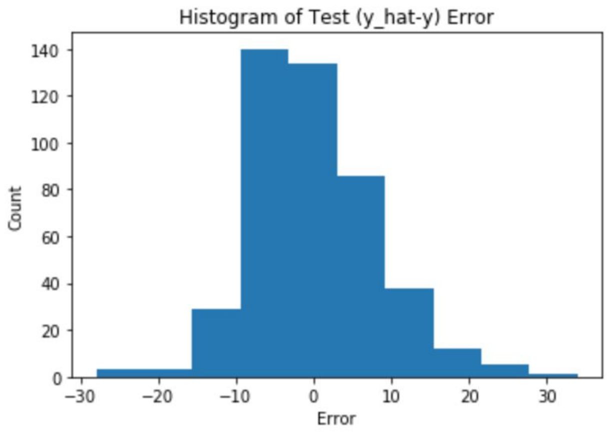
Album	Artist	Genre	Label	Producer	Release Date	Meta Score
Black and White Rainbows	Bush	['pop rock', 'alternative rock']	['caroline', 'zuma rock records']	['bob rock', 'gavin rossdale']	Mar 10 2017	49
Different Creatures	Circa Waves	['pop rock']	['virgin emi']	['alan moulder']	Mar 10 2017	78
Renditions	SECRETS	['post-hardcore']	['rise']	['beau burchell']	Apr 3 2015	NaN
Lost Property	Turin Brakes	['pop rock']	['cooking vinyl']	NaN	Jan 29 2016	78
What's Real	WATERS	['alternative rock']	[vagrant]	NaN	Apr 7 2015	NaN

III. Data Analysis

We had two main components to our data analysis step. First, we built a regression model to predict metacritic score based off the genres an album is listed under. An album can have one to several genres listed (e.g the album “Moh Lhean,Why?” has the genres “hip hop”,”indie pop”, and “rap”). Metacritic scores range from 0 to 100, and our final model had a training mean absolute error of 5.39 and a test mean absolute error of 6.25. By looking at the weights of the model, we can see which genres are associated with higher ratings (e.g. “grunge” is associated with higher ratings whereas “west coast hip hop” is associated with lower ratings).

Table 2: Top and lowest genre weights from regression model

Top Genre Weights		Lowest Genre Weights	
Genre	**Weight**	**Genre**	**Weight**
Gansta Rap	14.48	Electronic Music	-17.24
Stage & Screen	12.63	Holiday	-13.40
Post Metal	7.91	G Funk	-7.48
Grunge	7.32	Electro House	-7.15
Punk	7.23	West Coast Hip Hop	-7.06



Secondly, we did association rule mining to determine which genres tend to occur together. This step made use of all the data, including those that did not have metacritic scores. Using Table E, we first removed any tuples that did not contain any genres. This reduced the number of tuples from 5194 to 4972. Of the 4972 tuples, there were 475 unique genres. In order to get interpretable results, we only looked at genres that appeared four times or more.

The table below is a sample of the association rules we generated. Refer to the Stage 4 repository for complete association rules data (called *rules.csv*). Note that for genres “west coast hip hop” and “g funk,” our rule says that “west coast hip hop” is always included whenever “g funk” is (**confidenceBtoA** = 1). However, this does not tell us if there is a relationship between the two genres or if they are occurring together in the same genre list for an album simply by chance. To answer this question, we refer to **lift**, which takes into account the popularity of both genres. That is, for **lift(**“west coast hip hop”, “g funk”**)** = 1.717 > 1 implies that there is a positive relationship between the two genres (i.e. “west coast hip hop” and “g funk” occur together more often (1.717 times more) than random).

Table 3: Sample of generated association rules used to understand the relationship between any given pair of genres

item_A	**item_B**	**support AB**	**supportA**	**supportB**	**confidence AtoB**	**confidence BtoA**	**lift**
comedy	soundtrack	0.058	0.116	0.175	0.5	0.333	2.862
vocal jazz	big band	0.116	0.233	0.233	0.5	0.5	2.146
dub	idm	0.058	0.116	0.233	0.5	0.25	2.146
west coast hip hop	g funk	0.233	0.582	0.233	0.4	1	1.717
west coast hip hop	gangsta rap	0.349	0.582	0.0.582	0.6	0.6	1.030

By looking at both of our data artifacts, we can gain insight into which genres an album could realistically/truthfully list under (by using the association rules) that may be associated with higher critic ratings (by using our linear model). For example, an association rules states that “west coast hip hop” and “gangsta rap” tend to occur together. But the linear model says that “gangsta rap” genre (weight of 14.48) tends to be associated with high ratings whereas “west coast hip hop” (weight of -7.06) tends to be associating with low ratings. So if an album is about to be released, its publishers would prefer to use the “gangsta rap” genre listing.

One last common application for using association rule mining would be for recommender systems. Once genre pairs have been identified as having a positive relationship, recommendations can be made to customers. For example, it may introduce customers to new genres that they never would have listened to before or imagined existed.

IV. Future Work

In the future, we would want to try adding some additional features to the critic prediction model. We could add album name information (perhaps some words are associated with higher rating albums), as well as the month release date (e.g perhaps successful albums tend to launch in October for Christmas sales). Additionally, it would be interesting to add the sales associated with the albums, which may help uncover what artists and/or genres seem to be popular during certain periods.

We also might want to look at which labels and/or producers are the best for an album’s success, depending on its genre(s). This might require combining related genres into broader categories for easier analysis.

V. Code to Merge Tables (Next Page)

Merge Script

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

import warnings
warnings.filterwarnings(action="ignore", module="scipy", message="^internal gelsd")

# Load the pre-labeled data
metacriticData = em.read_csv_metadata("data/metacritic.csv")
wikiData = em.read_csv_metadata("data/wikiData.csv")

# add ID column to each dataset
metacriticID = ["a" + str(num) for num in np.arange(1, len(metacriticData.index)+1)]
wikiID = ["b" + str(num) for num in np.arange(1, len(wikiData.index)+1)]

col_idx = 0
metacriticData.insert(loc = col_idx, column = 'ID', value = metacriticID)
wikiData.insert(loc = col_idx, column = 'ID', value = wikiID)
em.set_key(wikiData, 'ID')
em.set_key(metacriticData, 'ID')

#read in labeled samples
S = em.read_csv_metadata("candidates_sample.csv",
                         key='_id',
                         ltable=metacriticData, rtable=wikiData,
                         fk_ltable='ltable_ID', fk_rtable='rtable_ID')

### CREATE LEARNER (from stage 3) ###

# Read in I and J
i_file = "I.csv"
j_file = "J.csv"
I = em.read_csv_metadata(i_file,key="_id",ltable=metacriticData,rtable=wikiData,fk_ltable="ltable_ID",
fk_rtable="rtable_ID")
J = em.read_csv_metadata(j_file,key="_id",ltable=metacriticData,rtable=wikiData,fk_ltable="ltable_ID",
fk_rtable="rtable_ID")

# Generate a set of features
F = em.get_features_for_matching(metacriticData, wikiData, validate_inferred_attr_types=False)

# Convert the I into a set of feature vectors using F
H = em.extract_feature_vecs(I,
                            feature_table=F,
                            attrs_after='label',
                            show_progress=False)

# create learners
random_state = 0

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

# Impute feature vectors with the mean of the column values.
H = em.impute_table(H,
                exclude_attrs=['_id', 'ltable_ID', 'rtable_ID', 'label'],
                strategy='mean')

#initial learning results
result = em.select_matcher([dt, rf, svm, ln, lg, nb], table=H,
        exclude_attrs=['_id', 'ltable_ID', 'rtable_ID', 'label'],
        k=5,
        target_attr='label', metric_to_select_matcher='f1', random_state=0)
result['cv_stats']

classifiers = np.array([dt, rf, svm, ln, lg, nb])

# Convert J into a set of feature vectors using F
L = em.extract_feature_vecs(J, feature_table=F,
                            attrs_after='label', show_progress=False)

# Impute feature vectors with the mean of the column values
L = em.impute_table(L,
                exclude_attrs=['_id', 'ltable_ID', 'rtable_ID', 'label'],
                strategy='mean')

for c in classifiers:
    # Train using feature vectors from I
    c.fit(table=H, exclude_attrs=['_id', 'ltable_ID', 'rtable_ID', 'label'], target_attr='label')

    # Predict on L (feature vector from J)
    predictions = c.predict(table=L, exclude_attrs=['_id', 'ltable_ID', 'rtable_ID', 'label'],
                            append=True, target_attr='predicted', inplace=False)

    predictions[['_id', 'ltable_ID', 'rtable_ID', 'predicted','label']].head()

    # Evaluate the predictions
    print(c.name)
    eval_result = em.eval_matches(predictions, 'label', 'predicted')
    em.print_eval_summary(eval_result)
    print()

### STAGE 4 ###

### FUNCTIONS###

# create function to remove char a/b in list of matches
def removeChar(string):
    return string[1:]

# use this function to edit metacritic genre column since Pop/Rock should be pop rock
def replaceBackslash(string):
    # make sure string is string
    string = str(string)

    # if empty, return the string
    if not string:
        return string

    # otherwise, replace backslash with a space
    return string.replace("/",' ')

def merge_genre_or_label_or_producer(v1, v2):
    # make sure string is string then set to lower case
    v1 = str.lower(str(v1))
    v2 = str.lower(str(v2))

    # remove brackets and single quotes for now
    v1 = v1[v1.find("[")+1:v1.find("]")].replace("'",'').replace("-",' ')
    v2 = v2[v2.find("[")+1:v2.find("]")].replace("'",'').replace("-",' ')

    # in case of multiple values for v1 or v2, we need to split by comma
    v1 = [x.strip() for x in v1.split(',')]
    v2 = [x.strip() for x in v2.split(',')]

    # combine v1 and v2
    v1.extend(v2)
    # remove anything in common
    v1 = list(set(v1))

    # remove any empty string
    v1 = [i for i in v1 if i]

    return v1

def merge_releaseDate(v1, v2):
    # always take wiki release date
    return v2

def merge_metaScore(v1, v2):
    # wiki has no meta score, so return metacritic score (v1)
    return v1

# this function takes two tuples and merges them
def mergeTuple(t1, t2):
    # create empty combined tuple (to add to table E)
    t = []
    for i in np.arange(1,len(t1)):
        if wikiData.columns[i] in ['Album', 'Artist']:
            if len(t1[i]) >= len(t2[i]):
                currT = t1[i]
                t.append(currT)
            else:
                currT = t2[i]
                t.append(currT)
        elif wikiData.columns[i] in ['Genre', 'Label', 'Producer']:
            currT = merge_genre_or_label_or_producer(t1[i], t2[i])
            t.append(currT)
        elif wikiData.columns[i] in ['Release Date']:
            currT = merge_releaseDate(t1[i], t2[i])
            t.append(currT)
        else:
            currT = merge_metaScore(t1[i], t2[i])
            t.append(currT)
    return t

### MERGE CODE ###

# read in complete candidate list
C = em.read_csv_metadata("candidates.csv",
                         key='_id',
                         ltable=metacriticData, rtable=wikiData,
                         fk_ltable='ltable_ID', fk_rtable='rtable_ID')

# Convert candidate C into a set of feature vectors using F
K = em.extract_feature_vecs(C,
                            feature_table=F,
                            show_progress=False)
# Impute feature vectors with the mean of the column values
K = em.impute_table(K,
                exclude_attrs=['_id', 'ltable_ID', 'rtable_ID'],
                strategy='mean')

# take best classifier (linear regression) and output predictions (i.e. matches)
predictions = ln.predict(table=K, exclude_attrs=['_id', 'ltable_ID', 'rtable_ID'],
                        append=True, target_attr='predicted', inplace=False)

# save set of matches between two tables
matches = predictions.loc[predictions['predicted'] == 1]
matches = C.loc[matches.loc[:,'_id'],:]
matches.to_csv("matches.csv", index = False)
```

```
# get a list of the indices to match
metacriticIndexMatches = [int(removeChar(s))-1 for s in list(predictions.loc[predictions['predicted'] ==
1,'ltable_ID'])]
wikiIndexMatches = [int(removeChar(s))-1 for s in list(predictions.loc[predictions['predicted'] == 1,'
rtable_ID'])]

# remove "/" in Genre column of metacriticData
metacriticData['Genre'] = metacriticData['Genre'].apply(lambda x: replaceBackslash(x))

# replace all NaNs with an empty string since that'll be easier for me to work with
metacriticData = metacriticData.replace(np.nan, '', regex = True)
wikiData = wikiData.replace(np.nan, '', regex = True)

# initialize table E that merges metacriticData and wikiData
E = pd.DataFrame(columns = metacriticData.columns[1:])

# add all matches
for i in np.arange(0,len(wikiIndexMatches)):
    # merge tuples in wikiIndexMatches and metacriticIndexMatches (exclude ID column)
    metaTuple = metacriticData.loc[metacriticIndexMatches[i]][0:]
    wikiTuple = wikiData.loc[wikiIndexMatches[i]][0:]
    mergedT = mergeTuple(metaTuple, wikiTuple)

    # add to E
    E.loc[i] = mergedT

# add everything NOT in metacriticIndexMatches and wikiIndexMatches
metaIndices = list(range(np.shape(metacriticData)[0]))
wikiIndices = list(range(np.shape(wikiData)[0]))
leftoverMetaIndices = [i for i in metaIndices if i not in metacriticIndexMatches]
leftoverWikiIndices = [i for i in wikiIndices if i not in wikiIndexMatches]

# subset data based on leftover indices (exclude ID column)
subsetMetaData = metacriticData.iloc[leftoverMetaIndices, 1:]
subsetWikiData = wikiData.iloc[leftoverWikiIndices, 1:]

# lower case genre, label, and producer cell values
toLowerCase = ['Genre', 'Label', 'Producer']
for col in toLowerCase:
    subsetMetaData[col] = subsetMetaData[col].str.lower().str.replace("-",' ')
    subsetWikiData[col] = subsetWikiData[col].str.lower().str.replace("-",' ')


# combine with E
E = pd.concat([E, subsetMetaData], axis = 0)
E = pd.concat([E, subsetWikiData], axis = 0)

# change index of E
E.index = range(np.shape(E)[0])

# replace empty strings and lists with NaN
E.replace(r'^\s*$', np.nan, regex=True, inplace = True)

#convert all elements to string first, and then compare with '[]'. Finally use mask function to mark
 '[]' as na
# then save to "E.csv"
E.mask(E.applymap(str).eq('[]')).to_csv("E.csv", index = False)
```