Movie Exercise Report

Rui Li
1/22/2018

Task:

This exercise focuses on **normalizing movie metadata and creating canonical versions of movies** originating from different sources.
The file Movie.json contains data in JSON format for movie references. The reference_id uniquely identifies a reference. Design a scheme to create canonical movies from these references and implement it in Python or Java.
The output for a canonical should be in the following JSON format:
{
“canonical_id”: …,
— this is an unique ID assigned by you for this canonical
“references”: []
— these are the reference_ids of entities that are deemed similar enough to belong to this cluster. An array of one or more reference_ids.
“title”: …,
— the “best” title for this canonical derived from the titles of contributing references
You get to define what “best” means. Similarly
“description”:
“content_rating”:
“genre”:
“release_date”:
“cast_and_crew_all”:
}
Output result: a json file that contains movie canonicals with clustered references.
Please provide a zip or tarball of your python/java code and output file and command line to reproduce the result.
For example, if you did the exercise in Java:
%java -cp “.” myPackage.foo.Mainclass -inFile inputFileName.json -outFile outputFile.json
Also include a brief write-up of the approach taken, design choices and decisions made.

Approach:

Step 1: create pojos for Movie, Movies and deserialize the input json file into a Movies object.

Step 2: run Movies.analyze() to analyze the data completeness of each field and visualize check the content of each field in Movie.

attribute	**Number of movies with this field**
title	913
description	860
release_date	669
release_year	912
run_time	892
role_director	889
role_actor	870
rating	751
genre	800

As you can see from the above table, some attributes are missing. Thus we need to handle missing attributes when we define similarity measures.

Step 3: extract/normalize different fields of all the Movie objects:

- Deserialize each "**person**" string into Person object
- I found there are movies with missing release_year but they have release_date. I also found that some release_date and release_year don't match. So in order to match the years of the movies, I extract years from both fields and put them into Movie.years field then later use that for similarity measure
- Genre looks very noisy. So I pulled the "complete" list of genres from IMDB to normalize each genre of the movie.

Step 4: Since the data is not labeled, so we can't do supervised solution classifier. So I will define a Similarily class to calculate the similarity of two Movie objects (A, B). In the class, there are separate similarities defined for different metadata. Then I will play with different combination of attribute similarity to come up with good measure. Once two movies are considered "same" by the measure, they are connected in the movie graph. Then I can use a union set based connected component algorithm to get the clusters of "same" movie.

attribute	**Similarity function**
title	Jaro Winkler distance
description	Do not use due to limited time
release_year	equal? 1 : 0
run_time	min(A.run_time, B.run_time)/max(A.run_time, B_run_time)
director	Is there intersection between two sets of directors ? 1 : 0
actor	Is there intersection between two sets of actors ? 1 : 0

The overall similarity is defined by Similarity.same() function:

```
public boolean same(double thresh){

  double s1 = (title + year)/2;
  double s2 = (title + actor + director)/3;
  double s3 = (year + actor + director)/3;

  return s1 >= thresh || s2 >= thresh || s3 >= thresh;
}
```

The principle is: one of the following 3 conditions is satisfied

- Title and year match
- Title, actor and director match (sometimes there is a new releases of the movie after a few years, but they should be considered as same movie)
- Year, actor and director match (based on assumption that one set of actors + directors can only make 1 movie per year)

Step 5: generate the canonical form of a movie cluster
Here is my simple solution to generate each attribute:

attribute	Canonical form
canonical_id	Incremental from 1-107
title	Majority voting
description	Majority voting
release_year	Majority voting of the extracted “years” field
rating	Dedupped combination of all ratings
genre	Dedupped combination of all normalized genres
run_time	average
cast_and_crew_all	Union of all role_director from each movie
role_director	Union of all role_director from each movie
role_actor	Union of all role_actor from each movie

Experiments and tuning：

Step 1:

The deserialization gives 913 movies.

Step 2:

After analyzing the attributes and eye balling the values, I have made the following operations on the data:

- Normalize the title before calculating title distance
- Get an official list of Genre from IMDB and only accept the genre values in that list.

Step 3:

- Criteria 1: title similarity = 1.0 and release_year similarity = 1.0. → 336 clusters
- Criteria 2: combination of various similarities: → **107 clusters (my final output)**

Performance evaluation:

Since the data is not labeled, it’s hard to have a quantitative measure of the performance. I could only visually check the cluster quality. It looks not bad. It clustered 913 movies into 107 clusters. It can cluster the same movie from different languages to the same cluster. It can also handle the cases where some fields are missing (year, person’s character name, etc). The genre field of the canonical form is now very clean due to the normalization.

Future work:

- I noticed that some movies should have been clustered into another cluster but they do not. I checked the data. It's mainly caused by different language version of animation movies (Despicable Me, Inside Out, etc). The none-English version often has the actor's name to be the person who gives voices in their language and the character's name is often displayed in their own language. So in order to handle such cases, we need to add more rules in Person matching and probably leverage some machine translation library.
- Due to the nature of the unlabeled data, I can only do unsupervised clustering and come up a similarity rule by common sense. If I had these data labeled, I could train a LR model to automatic decide on the weight of each field.

How to run the package:

Everything (jar, src, input data, output data) is packaged in the MovieExercise_RuiLi.tar.gz including a pdf of this file.

Command (also packed into _run.sh in target folder):
java -cp target/movie-1.0-SNAPSHOT-jar-with-dependencies.jar Movies
/Users/ruili1/Downloads/movies_exercise/movies.json
/Users/ruili1/Downloads/movies_exercise/output.json

output:

913 movies in /Users/ruili1/Downloads/movies_exercise/movies.json
107 canonical forms saved to /Users/ruili1/Downloads/movies_exercise/output.json