Movie Data Analysis Using Vector Models

Jingyang Guo

Group Members

Yang Jianan Yang Mingfei Yang Rui
Li Chenyang Matlani Sagar

Abstract

In phase 1 of the course project, we are given a set of movie data focusing on tags attached to movies by users. We are asked to use vector models to calculate weights of each tag for actors, genres and users, while tag time and actor rank are considered in the models. We are also asked to applied different models to differentiate from two genres using their tag vectors. I use Python throughout this project. And this report introduces how I reached the goals above.

keywords: vector model, TF-IDF, Python

Goal Description

1. Create weighted tag vectors for each actor using TF and TF-IDF model.
2. Create weighted tag vectors for each genre using TF and TF-IDF model.
3. Create weighted tag vectors for each user using TF and TF-IDF model.
4. Explain in what ways one genre differs from another considering three models: TF-IDF-DIFF, P-DIFF1 and P-DIFF2.

Solution

For the first three goals, I used similar approaches. First I find all movies related to the actor/genre/user, and then find all tags and their weights related to those movies.

For the last goal, I calculate the vector of the first genre using the model specified in the input, and output the result of vector1. In the result, the tags with a larger weight are where genre1 differ more from genre2.

Implementation

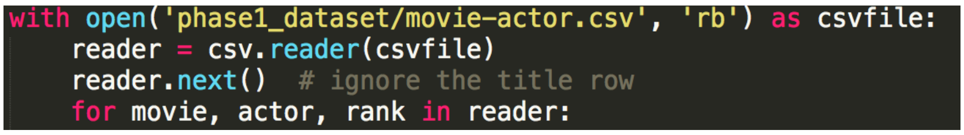
I created a single file using Python 2.7 to finish this project. The follow steps describes how I did it in detail.

1. Command line interface

I use Python's native library "argparse" to parse command line input. This library makes it convenient to create a command line interface with help information and input constraints. The code to set up the parser is in line 386-405 in "movie.py".

2. Reading the csv files

I create a getter function for each file. This function reads the csv file using the following code, and store it in a global "list" or "dictionary" variable for future use.



```python
with open('phase1_dataset/movie-actor.csv', 'rb') as csvfile:
    reader = csv.reader(csvfile)
    reader.next()  # ignore the title row
    for movie, actor, rank in reader:
```

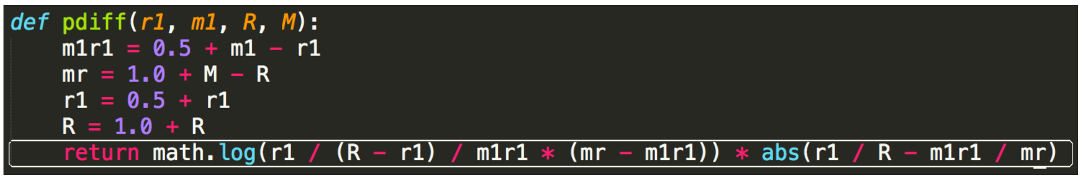
When reading "mltags.csv", the function also calculate the time weight for each row. It converts the time string to unix timestamp(seconds passed since 1-1-1970), and map the timestamp to range of 0.5 - 1 as the weight.

3. Calculating the vector

For TF model , I first go through the data to find all movies related to the specified actor/genre/user, and then go through "mltags.csv" to find tags and weights related to the movies. For actor, the rank in a movie is also considered as weight. In my code, I map rank 1 - 20 to weight of 1 - 0.4, and rank > 20 to weight of 0.4.

For IDF model, I additional create a dictionary mapping tag to actor/genre/user when go through the data, in order to calculate the idf value for each tag.

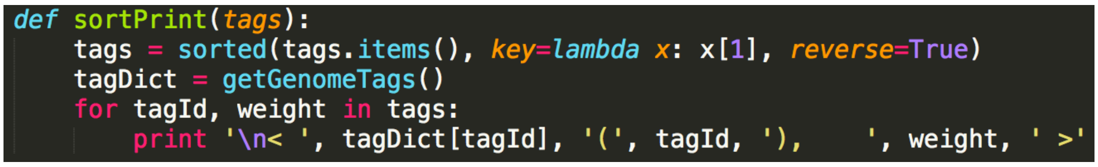
For P-DIFF model, I mainly use "set" to store related movies, so the length of the sets are the values of M, R, m1 and r1. Calculation of the weight is in "pdiff" function. To prevent 0 from showing in log or being a divisor, I add 1 or 0.5 to the parameters as shown below.



```python
def pdiff(r1, m1, R, M):
    m1r1 = 0.5 + m1 - r1
    mr = 1.0 + M - R
    r1 = 0.5 + r1
    R = 1.0 + R
    return math.log(r1 / (R - r1) / m1r1 * (mr - m1r1)) * abs(r1 / R - m1r1 / mr)
```

4. Outputting the result

Sorting and outputing are implemented in "sortPrint" function. It sorts the tags in a descending order, then checks the "genome-tags.csv" to get names of the tags, and print all tags to the terminal window.



```python
def sortPrint(tags):
    tags = sorted(tags.items(), key=lambda x: x[1], reverse=True)
    tagDict = getGenomeTags()
    for tagId, weight in tags:
        print '\n< ', tagDict[tagId], '(', tagId, '),    ', weight, ' >'
```

Interface specification

My code requires Python 2.7 installed.

In the terminal window, "cd" to the "Code" folder, and type "python movie.py" to run my code.

Then you can input the following 4 kinds of command for any times. The output will show tags with higher weights first.

print_actor_vector actorId model(TF or TF-IDF)

print_genre_vector genreName model(TF or TF-IDF)

print_user_vector userId model(TF or TF-IDF)

differentiate_genre genre1 genre2 model(TF-IDF-DIFF, P-DIFF1 or P-DIFF2)