CSE 515 Group Project

Phase #1 Report

Group 7 – Mingfei Yang

Other Members – Sagar Matlani, Jingyang Guo, Chenyang Li, Jianan Yang, Rui Yang

Abstract

The processing program can build different tag-weight vectors under different models according to the input commands and output the vectors to csv files. When constructing tag-weight vectors under TF model, raw count of each tag is applied as its raw term frequency measurement, and its relevant time weights (as well as rank weights when the vector is for actors) are combined onto the raw term frequency values, finally get the combined tag-weight vectors as the results. When constructing under TF-IDF model, combined-weighted TF values are calculated in the same way as TF model, for IDF values, the $\log\frac{N}{nt}$ is used, where N is the entire set of actors, genres, or users, n_t represents the number of tag t's existences in the entire set N. For differentiating two genres, under TF-IDF-DIFF model, the tag-weight vectors of genre 1 and genre 2 are built respectively; under P-DIFF model, the weight vectors are computed under given formula, and boundary cases are properly handled.

General Keywords and Phrases: Raw count, time-weighted, inverse frequency, discrimination, probabilistic relevance, feature significance

Introduction

Terminology

- TF, abbreviation of Term frequency, a statistic that measures how many times a term appears in the document, the simplest choice is to use the raw count of a term, denoted as $f_{t,d}$ [1]. The measurement used in this phase is the raw count.
- IDF, abbreviation of Inverse Document Frequency, a statistic that measures how important a term is to a document in a collection or corpus [1]. There are also multiple choices to measure IDF value, in this phase, the $\log\frac{N}{nt}$ is applied, where N represents the collection or corpus of all documents, n_t represents the number of documents in which feature t (tag t in this project) exists.
- TF-IDF, the combination of two statistics above. It is measured as TF value multiply the IDF value.
- Time-weighted, all the resulting TF values and TF-IDF values are time-weighted due to each tag may appear multiple times, thus it may have multiple timestamps.
- Rank-weighted, the resulting TF values and TF-IDF values for actor-tag vector space are also rank-weighted due to each tag may be mapped to multiple actors, but actors as major characters are better-represented by this tag.
- Probabilistic Relevance Feedback, a mechanism of measuring feature significance based on the relevance information given by users [2].

Goal Description

The phase 1 is divided into 4 tasks. The goal of task 1 is to construct a time-weighted and rank-weighted actor-tag vector for given actor ID under TF model or TF-IDF model. In task 1, the corpus of documents is the actor information data set, i.e. all the unique actors in the database. The goal of task 2 is to build a time-weighted genre-tag vector for given genre under TF model or TF-IDF model. In task 2, the corpus of documents should be all the unique genres in the database, but in the given data sets, there is no separate genre set, so the corpus is built on-the-fly by traversing all movies' genres and put all unique genres into a list. The goal of task 3 is to obtain a time-weighted user-tag vector for given user ID under TF or TF-IDF model. In task 3, the corpus of documents is the entire users set. The goal of task 4 is to differentiate two given genres under two kinds of model, one is the time-weighted TF-IDF model, in this model, the corpus of documents is changed; the other is probabilistic relevance feedback model, with two ways of calculating the weight vector.

Assumptions

1. TF values are time-weighted (and rank-weighted for actor-tag vectors), but the IDF values are not involved with timestamps. Thus, the weighted TF-IDF values are computed by weighted TF values multiplying raw IDF values. Both TF and TF-IDF values are not normalized to certain range.
2. In task 4, differentiating two genres under TF-IDF-DIFF model, the document is assumed to be the genres. But corpus of genres differs from the corpus of task 2: the corpus is not the entire set of genres, but the set of genres related to all the movies in the given genre 1 or the given genre 2.
3. In task 4, differentiating two genres under P-DIFF model, in order to prevent the value inside logarithm to be 0, proper approximations are applied for extreme or boundary cases [2][3]:

   $$p(f|Relevant) = \frac{r+0.5}{|R|+1} \qquad p(f|Irrelevant) = \frac{(m-r)+0.5}{|M-R|+1}$$

   where r, m R, M have the same meanings annotated in project specification.

Solution and Implementation

The processing program is implemented in Python, and each task is implemented in separate python files. Although this will cause duplicated code, the logic of solution and implementation behind each task will be clearer.

For task 1, the first step is count the total number of documents, which is the total number of actors for this task, by scanning the given data set of actors' information. Then open the data set file which contains the actor ID, movie ID, and actor rank, to capture all the relevant movies and ranks for the input actor ID. After obtaining the relevant ranks, the next step is to convert the ranks in proper way so that we can combine it to the weight of tags. The rank normalization is done by 2 steps, first step is to find the highest value of rank, let's say r(high), and the lowest value of rank, say r(low), then compute the difference as r(high) – r(low). The second step is to normalize every rank in the range of 0 to 1, by applying this formula to each rank:

$$weight(r) = \frac{r(high) - r}{r(high) - r(low)}$$

The weight of low rank values (actors are major characters) will have the value close to 1, and the weight of high rank values will be close to 0. The weight of certain rank of the given actor is added up to the weight of associated tags. As for relevant movie IDs, they are used to find out all relevant tag ID and timestamp pairs in the data set file. The normalization of timestamps is similar to the rank normalization, find the newest timestamp, say ts(new), and the oldest timestamp, say ts(old), then normalize all timestamps in the range of 0 to 1 using the formula:

$$weight(timestamp) = \frac{ts - ts(old)}{ts(new) - ts(old)}$$

The weight of new timestamps will have the value close to 1, and the weight of old timestamp will be close to 0. And the weight of certain timestamp is added up to the weight of its associated tag in the tag-timestamp pairs. Under TF model, the raw count of each tag is applied to represent its raw TF value. For each tag-timestamp pair, the weighted TF value is 1 + weight(timestamp) + weight(rank), where the weight(rank) is decided by the associated movie ID of this tag-timestamp pair. If the tag appears multiple times, which means it will have multiple timestamps and ranks, the final weighted TF value will be the sum of each weighted value of each pair. Under this scheme, each tag-timestamp pair's combined weight will be in the range of 1 to 3, so if a tag appears k times, then the final weight of the tag will be in range of k to 3k. Under TF-IDF model, the TF values are calculated in the same way as in TF model. The IDF values are computed without timestamps and ranks. It simply measures the inverse frequency of actors with certain tag. The N inside the logarithm is the total number of actors, while the n_t is the number of actors who associate with tag t.

For Task 2 and task 3, solution and implementation are similar to task 1, with minor modifications. No ranks involved in task 2 and task 3, so the weighted TF values are only time-weighted. And the corpus of documents in task 2 is also slightly complicated than task 1 and task 3. Because there is no specific genre data set, so the corpus of genres is obtained by traversing all the genres associated with every movie. The TF values are computed in the same way, for each tag-timestamp pair, the weighted value is 1 + weight(timestamp). The IDF values measure the inverse frequency of genres or users with certain tag.

For task 4, although the solution is related to genre-tag vectors as in task 2, the corpus is different. Under TF-IDF-DIFF model, the main idea of calculation process remains the same, but the number of all documents is narrowed to the subset that related to two given genres. Under two P-DIFF models, there are variances in the definition of statistics involved in the probabilistic relevance feedback formula, but the main process is still similar. For extreme and boundary cases, proper approximations [2][3] are applied to avoid 0 value inside logarithm as well as the denominator.

For P-DIFF models in task 4, the larger the output weight for certain tag t, the more relevant the tag t to input genre 1. Despite of doing approximation in some cases, there are some other situations that the program will not do the calculation, instead, for those particular tags, specific explanation is given. I briefly explain such situations below.
In P-DIFF-1 model, where r represents the number of movies which contain tag t in input genre 1, m represents the number of movies which contain tag t in input genre 1 or input genre 2, R represents the total number of movies in genre 1, M represents the total number of movies in genre 1 or genre 2. For two input genres, if they are the same, or they produce the same value of M and R, then no calculation will be done, instead, the program will directly output that the two given genres have no discrimination because they produce the same set of movies. Same set of movies will have same set of tags. And there is also another situation that for a particular tag t, the m is equal to r, in P-DIFF-1 model, it means that all the movies that contain the tag t are in genre 1. In the case, no calculation will be done either, the program will mark the tag t as "only relevant to Genre 1".
In P-DIFF-2 model, where the definition of r, m and R changed, r represents the number of movies which don't contain tag t in input genre 2, m represents the number of movies which don't contain the tag t in input genre 1 or input genre 2, R represents the total number of movies in genre 2. For two input genres, if they are the same, or they produce the same value of M and R, it will be handled in the same way as in P-DIFF-1. For the situation that a particular tag t, the r is the same as R, in P-DIFF-2 model, it means that the tag t is irrelevant to all the movies in genre 2, which in turn, indicates that the tag t only appears in movies in genre 1. In this case, no calculation will be done, the program will mark the tag t as "only relevant to Genre 1".

Interface Specification

The functions which process and generate vectors for different purpose are implemented in separate python file, but are accessed through the python file called FunctionCaller.py. More specifically, the functions are called by typing in certain commands as defined in project specification. For actor-tag, genre-tag, and user-tag vectors, the commands are in the following format:

```
print_actor_vector actorID model
print_genre_vector genre model
print_user_vector userID model
```

where the first part will locate to certain functioning python file, second part is used as input, and third part is used as computing scheme.
For differentiating two genres, the command is in the following format:

```
differentiate_genre genre1 genre2 model
```

where the first part will locate to certain functioning python file, second and third parts are used as input genres, and the last part indicates the computing scheme.

System Requirements, Installation and Execution

The program is implemented under Python 3.6.2 environment with PyCharm as the IDE. Thus, Python should be pre-installed on the user's environment (users can download Python for their own platform through https://www.python.org/downloads/, and Python 3.6.2 is recommended). Installation of the process program is not required because the python files can be accessed directly through command line as well as IDEs.
Here are the two ways to execute the program in the following environment.
System Environment: macOS Sierra 10.12.6
Python Environment: Python 3.6.2

1. For command line access, users should open terminal or cmd in the "Code" repository or direct to that repository. Then type the command:

   python3 FunctionCaller.py

   The python file should be executed and will display as figure 1. Type "help" or "?" will display the command format, type "exit" or "q" will quit the application. For specific purpose or task, user needs to type in the correct command, valid input, and valid computing model (the program can handle some invalid cases and give feedback though). After executing certain command, the program will output the resulting vector both in the terminal and into a csv file inside the "Outputs" repository in the parent directory of "Code".
   Note: make sure the "Code" repository, "Outputs" repository and "phase1_dataset" repository exist and in the same level (have the same parent repository), and the data sets are put in the "phase1_dataset" repository.

```
Mingfei-MacBook-Pro:CSE515_Phase-1 myang$ cd Code
Mingfei-MacBook-Pro:Code myang$ python3 FunctionCaller.py
Type 'help' or '?' to List Available Commands
Type 'exit' or 'q' to Quit The Application
-->
```

figure 1 Terminal Access

2. For IDE access, users should pre-install certain IDE, for example, PyCharm (It can be downloaded through https://www.jetbrains.com/pycharm/download/). When open it on IDE, user needs to make sure that "Code", "Outputs" repository, and "phase1_dataset" repository (all data sets included) are under the same root repository. Then open this root repository in the PyCharm, and mark the "Code" repository as source root. The structure should be similar as figure 2 shows.

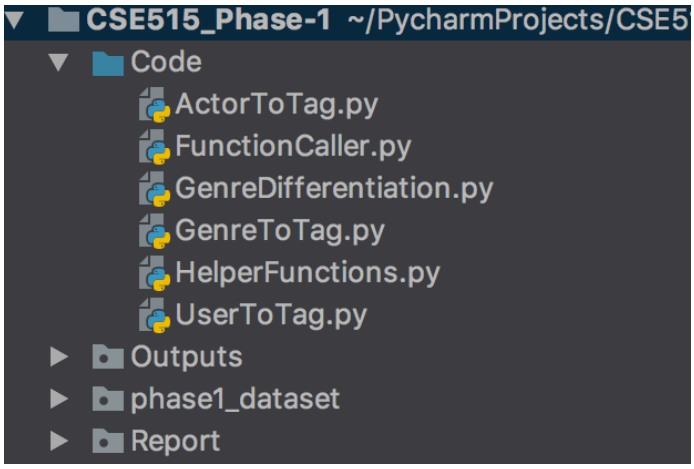

figure 2 Tree Structure in IDE

Then run the "FunctionCaller.py", it should automatically compile these python files and display the interface like figure 3.

```
/Library/Frameworks/Python.framework/Versions/3.6/bin/python3.6 /Users/myang/PycharmProjects/CSE515_Phase-1/Code/FunctionCaller.py
Type 'help' or '?' to List Available Commands
Type 'exit' or 'q' to Quit The Application
-->
```

figure 3 Interface of the Application

Related Works

Read related books [2][3] to find proper approximation method for probabilistic relevance feedback mechanism when encountering extreme and boundary cases.

Conclusions

With different computing schemes, users can get different views of how tags are relevant to given subjects (actors, genres, or users). The vector obtained for the same subject but under different model may not be the same. With IDF values involved, the weight of very common tags would be lower due to their generality, so more specific and less common tags would stand out to discriminate the subject from others. Probabilistic relevance feedback scheme can intuitively distinct two subjects from each other. If the result for certain tag is negative, it indicates that this tag (feature) describes the object better than the subject (genre 1 is the subject and genre 2 is the object in this phase). If the result for certain tag is positive and large, it indicates that this tag (feature) can efficiently differentiate the subject and the object.

The generated results also have some shortcomings. Since the TF values and TF-IDF values are not normalized, although it is still the larger the weight value, the higher the relevance of tag to certain subject (actors, genres, or users), it may cause some difficulties to measure the scale of the difference between tags.

Bibliography

[1] tf-idf – Definition – Term frequency. Wikipedia. https://en.wikipedia.org/wiki/Tf%E2%80%93idf
[2] C. D. Manning, P. Raghavan and H. Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press. Chapter 9, Section 9.1.2, pp. 183, Chapter 11, pp. 219 - 235.
[3] K. S. Candan, M. L. Sapino. 2010. Data Management for Multimedia Retrieval. Cambridge University Press. Chapter 12, Section 12.4.4, pp. 408

Appendix

This phase involves more individual works. But several discussions were held to help all group members to get better understanding and comprehension to the problem specification, as well as to reach some agreements on the tools, language, and interface for future possible code integration. Report review were also held to have basic understandings of each other's works, ideas, and conclusions.