CSE515 Multimedia and Web Databases
Report Phase#1
Group 7

Group member

Jianan Yang, Feiming Yang, Sagar Matlani, Rui Yang, Chenyang Li, Jingyang Guo

Abstract

Phase 1 targets on building tag vectors on the given data of MovieLens and IMDB. There are 4 tasks in phase1. For the first three tasks, a <tag, weight> vector should be generated given field id and vector model to measure how good is the tags for the given field as features. For the last task, a <tag, weight> vector is built to tell the difference between genre1 and genre2 with three vector models, which are TF-IDF-DIFF, P-DIFF1 and P-DIFF2.

Keywords

Vector Model, TF, TFIDF, TF-IDF-DIFF, P-DIFF1, P-DIFF2

I. Introduction

Terminology

- TF (Term frequency), is used to measure the frequency a specific term appearing in a document[1].
- $TF = \frac{\#\ term\ appears\ in\ the\ document}{\#all\ the\ terms\ in\ the\ document}$
- TFIDF is the multiply TF and IDF. IDF is inverse document frequency, which measures how much the term can discriminate the document from others.[1]

$$IDF_t = \exp(\frac{\#all\ the\ documents}{\#of\ documnens\ with\ term\ t\ in\ it})$$

- $TFIDF_j = TF_j \times normaliz(IDF_j)$
- TF-IDF-DIFF model shows the difference between two given genres. It's use modified IDF to compute TF-IDF. The difference in computing IDF is the number of documents is the number of all the unique genres which belongs to $movie(g_1) \cup movie(g_2)$ instead of the number of unique genres all the movies have.
- P-DIFF1 model shows the difference between two given genres. [3]

$$w_{1,j} = \log(\frac{(r_{1,j}/R - r_{1,j})}{(m_{1,j} - r_{1,j})/(M - m_{1,j} - R + r_{1,j})}) \left|\frac{r_{1,j}}{R} - \frac{(m_{1,j} - r_{1,j})}{M - R}\right|$$

  $r_{1,j}$ is the number of movies in genre, g_1, containing the tag t_j.
  $m_{1,j}$is the number of movies in genre, g_1 or g_2, containing the tag t_j.
  R is the number of all the movies containing g_1.
  M is the number of all the movies containing $g_1\ and\ g_2$
- P-DIFF2 model shows the difference between two given genres. [3] The difference from P-DIFF1 is the definition of $r_{1,j}$ and $m_{1,j}$.

$$w_{1,j} = \log(\frac{(r_{1,j}/R - r_{1,j})}{(m_{1,j} - r_{1,j})/(M - m_{1,j} - R + r_{1,j})}) \left|\frac{r_{1,j}}{R} - \frac{(m_{1,j} - r_{1,j})}{M - R}\right|$$

  $r_{1,j}$ is the number of movies in genre, g_1, **not** containing the tag t_j.
  $m_{1,j}$is the number of movies in genre, g_1 or g_2, **not** containing the tag t_j.
  R is the number of all the movies containing g_1.
  M is the number of all the movies containing $g_1\ and\ g_2$

Goal description

The goal of phase1 is to experiment with vector models, by building <tag, weight> vectors on MovieLens and IMDB database with model TF and TF-IDF. Another goal is to learn how to differentiate genre1 and genre2 by three TF-IDF-DIFF, PDIFF1 and PDIFF2.

Assumption

1. In task1, assume all the actors in the given database forms whole documents, whether the actor's movie has been given a tag or not. In this case, there will be some actors who do not have any tags.
2. In task2, assume all the genres have at least one movie and at least one tag.
3. In task3, assume all the movies watched by a user is the set of movies the user gives a tag or a rating, even though some user may not give tag nor rating.
4. Assume for all the movies, movieid, tagid and timestamps forms a unique tag.

II. Implementation

Task1

For actors, according to the rule that tags with newer timestamp and higher actor rank should be given higher weight in terms of TF. Thus for a specified actor with actorid, find all the movies the actor participate in and find all the tags those movies have. Then form a list of tuples, where the tuple includes actor_rank, tagid, timestamp.
Actorid:[(actor_rank, tagid, timestamp)], here I use python style where '[]' means a list, and '()' means tuple.
Each tuple forms a **unique tag**, sort the list in descending order by timestamp and rank respectively. Then for the sorted list, compute the weight for tags in the list. Both timestamp and rank use the following formula.

$$W_j = \frac{\sum_{All\ the\ unique\ tags\ whose\ tag\ id\ is\ j} Index_j}{\sum_{All\ the\ unique\ tags\ in\ the\ list} Index}$$

,where $W_j\ means\ the\ weight\ for\ the\ tag\ whose\ tag\ id\ is\ j$, $Index_j$ means the index of the unique tag whose tagid is j, $Index$ is the index of the unique tag in the sorted list.
Here is the reason why W_j is the same as TF (term frequency) by definition,

$$TF = \frac{\#\ term\ appears\ in\ the\ document}{\#all\ the\ terms\ in\ the\ document}$$

Since we have got W_jTS (tag weight for timestamp) and W_jRank (tag weight for actor rank), I simply do summation of W_jTS and W_jRank and then do normalization. Then the final TF is

$$TFj = normalize(W_jTS + W_jRank)$$

The normalization given a list X used in this project is defined as

$$N_j = \frac{X_j}{\max(X)}$$

,where X is the whole list, $X_j\ is\ the\ item\ with\ index\ j, N_j\ is\ the\ normalized\ value\ for\ X_j$

Then compute IDF for term t.

$$IDF_t = \exp(\frac{\#all\ the\ documents}{\#of\ documnens\ with\ term\ t\ in\ it})$$

The document here is the actor. So the number of all the documents here is the number of all the unique actors who participate in the at least a movie. And for a specific actor with actorid, get a list of tagids, which is all the tags in the movies this actor participate in, and count how many actors contain the tags in the list.
Finally compute TF-IDF

$$TFIDF_j = TF_j \times normaliz(IDF_j)$$

Task2

For genres, according to the rule that tags with newer timestamp higher weight in terms of TF. Thus for a specified genre with genre_name, find all the movies which have this genre_name and find all the tags those movies have. Then form a list of tuples, where the tuple includes tagid, timestamp.
Genre_name:[(tagid, timestamp)], here I use python style where '[]' means a list, and '()' means tuple.
Then the method for computing TF and TF-IDF is the same as task1.

Task3

For users, according to the rule that tags with newer timestamp higher weight in terms of TF. Thus for a specified user with userid, find all the movies which this user have rated or tagged and find all the tags those movies have. Then form a list of tuples, where the tuple includes tagid, timestamp.
Userid:[(tagid, timestamp)], here I use python style where '[]' means a list, and '()' means tuple.
Then the method for computing TF and TF-IDF is the same as task1.

Task4

- TF-IDF-DIFF:
  The implementation of TF-IDF-DIFF is very similar to task 2. The only change is when computing IDF, the number of documents is the number of all the unique genres which belongs to $movie(g_1) \cup movie(g_2)$, instead the number of unique genres which all the movies have.
- P-DIFF1 and P-DIFF2

$$w_{1,j} = \log\left(\frac{r_{1,j}/(R - r_{1,j})}{(m_{1,j} - r_{1,j})/(M - m_{1,j} - R + r_{1,j})}\right)\left|\frac{r_{1,j}}{R} - \frac{(m_{1,j} - r_{1,j})}{M - R}\right|$$

  ,where R is the number of all the movies with genre 1, and M is the number of all the movies contains genre1 and genre2.
  The difference between P-DIFF1 and P-DIFF2 lies in the definition of $r_{1,j}\ and\ m_{1,j}$. In P-DIFF1, $r_{1,j}$ denotes the number of movies in genre, g_1, containing tag t_j, $m_{1,j}$ denotes the number of movies in genre, g_1 or g_2, containing tag tj. While in P-DIFF1, $r_{1,j}$ denotes the number of movies in genre, g_1, not containing tag t_j, $m_{1,j}$ denotes the number of movies in genre, g_1 or g_2, not containing tag t_j.
  To implement PDIFF, first build a dictionary of list whose key is the movieid, and the value is a list of genres corresponding to the movieid, like {movieid:[genre name]}. Then we can get R and M from this dictionary by counting the number of movies containing g1 or containing g1 or g2.
  Next, for g_1 and g_2, build a dictionary whose key is movieid and value is a list of tags. The tags must be the ones of with the movieid, like g1:{movieid:[tagid}}. Thus we can count the number of movies containing or not containing tag t_j for g_1 or g_2.
  Finally, use the weight formula to compute the weight.
  To avoid 0 in the denominator, the $P(f_j|R)\ and\ P(f_j|I)$ can be estimated as follows[2].

$$P(f_j|R) = \frac{r_j + m_j/M}{R + 1},\ \ P(f_j|R) = \frac{(m_j - r_j) + m_j/M}{M - R + 1}$$

  Thus the weight formula change to as the following.

$$w_{1,j} = \log\left(\frac{(r_{1,j} + m_j/M)\times(M - m_{1,j} - R + r_{1,j} + m_j/M)}{(m_{1,j} - r_{1,j} + 1)\times(R - r_{1,j} + 1)}\right)\left|\frac{r_{1,j} + m_{1,j}/M}{R + 1} - \frac{(m_{1,j} - r_{1,j} + m_{1,j}/M)}{M - R + 1}\right|$$

III. Interface specifications

Input with a file

Sample input:

python src/phase1.py testcase.txt

The following commands are in the testcase.txt:

print_actor_vector 1484 TF-IDF
print_actor_vector 1484 TF
print_genre_vector Western TF
print_genre_vector Western TF-IDF
print_user_vector 146 TF
print_user_vector 146 TF-IDF
differentiate_genre Thriller Horror TF-IDF-DIFF
differentiate_genre Thriller Horror P-DIFF1
differentiate_genre Thriller Horror P-DIFF2

The sample output for this test file is in Out/output.txt.

Input with command

- Task 1:
  Sample input:
  python src/phase1.py print_actor_vector 1484 TF-IDF
- Task 2:
  Sample input:
  python src/phase1.py print_genre_vector Western TF-IDF
- Task 3:
  Sample input:
  python src/phase1.py print_user_vector 146 TF-IDF
- Task 4:
  Sample input:
  python src/phase1.py differentiate_genre Western IMAX P-DIFF2

IV. Installation and execution instructions

This project is under **Python 3.6** with dependencies pandas and numpy. Please note the project will not work under **Python 2.7.**
After the environment is setup, go to Code director, and run

python src/phase1.py command.

"**command**" can be a single command or a txt file of command as shown in the sample input in section V. All commands should be in the format given in the project description. The code is tested on windows 10 and Ubuntu 14, but not tested on Mac OS.

V. Related Works

Read the chapter 12 in text book "Data Management for Multimedia Retrieval" to figure out the probabilistic feedback mechanism. Besides, this is my first doing project with python. I spent serval hours on learning pandas and numpy.

VI. Conclusions

In this project, I implemented python program building TF and TF-IDF vector for tags given by actorid, genres and userid for dataset MovieLens and IMDB. Besides, I implemented TF-IDF-DIFF, PDIFF1, PDIFF2 vector model to differentiate two genres.

By analyzing the result of Task1 through Task3, the difference between TF and TF-IDF is shown clearly. Some tags ranks higher in TF and vice versa. After diving into raw data, tags which have a higher rank tend to appear many times in a document and in many different times. Such tags have very weak ability to differentiate documents. In this case, TF-IDF is a better model than TF.

The result of Task 4 shows the difference between genre1 and genre2. For example, the tag which has a higher rank indicates it more related to genre1. That means we use the rank of tags to differentiate genres.

VII. Bibliography

[1] G. Salton and C. Buckley. "Term-weighting approaches in automatic text retrieval". Information Processing & Management, 24 (5). 1988.

[2] K. Selçuk Candan, Maria Luisa Sapino. “Data Management for Multimedia Retrieval”. Cambridge University Press. 2010.

[3] K. Selçuk Candan. “project1f17”. 2017.

VIII. Appendix

This project is done by group members individually. We had serval meetings to discuss how to implement the project. Also we build an online discussion group to help each other.