CSE515 Multimedia and Web Databases
Report Phase#1
Group 7

Group member

Jianan Yang, Fei Ming, 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 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, g1, containing tag tj, $m_{1,j}$ denotes the number of movies in genre, g1 or g2, containing tag tj. While in P-DIFF1, $r_{1,j}$ denotes the number of movies in genre, g1, not containing tag tj, $m_{1,j}$ denotes the number of movies in genre, g1 or g2, not containing tag tj,.
First build a dictionary of list whose key is the movieid, and the value is a list of genres this movie has, 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.
Then build a dictionary whose key is movieid and value is a list of tags in this movie for g1 and g2 respectively, like g1:{movieid:[tagid}}. Thus we can count the number of movies containing or not containing tag tj for g1 or g2.
Finally, use the weight formula to compute the weight.
To avoid 0 in the denominator, simply add 1 to all the dominators.

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 P-DIFF2

Sample output:

```
(testenv) D:\asu\asu\CS\515\project\phase1>python src\phase1.py testcase.txt
actor_id : 1484, model : TF-IDF
, format : <tag_id, tag_name, weight>

                tag_id                tag_name                weight
                   998                survival      0.41677855711711914
                   673       mountain climbing      0.27442745892898673
                   852               ridiculous      0.10898882810014357
                   883                   scenic      0.09917557445844806
                   312               disturbing      0.09068321755185776
                  1059                   unique      0.00994636384344459
                  1047              true story    5.701721125108135e-18

actor_id : 1484, model : TF
, format : <tag_id, tag_name, weight>

                tag_id                tag_name                weight
                   998                survival      0.41677855711711914
                   673       mountain climbing      0.27442745892898673
                   852               ridiculous      0.10898882810014357
                   883                   scenic      0.09917557445844806
                   312               disturbing      0.09068321755185776
                  1059                   unique      0.00994636384344459
                  1047              true story    5.701721125108135e-18

genre_id : Western, model : TF
, format : <tag_id, tag_name, weight>

                tag_id                tag_name                weight
                   178                  canada      0.2932757726712511
                  1007          talking animals      0.2159043379324899
                   277            culture clash      0.14502867356540064
                   523                   horses      0.143562987203365
                   327               dreamworks      0.10044061924617971
                   310  disney animated feature      0.09566904433935668
                   309                  disney     0.006054988320263475
                   509                 history    6.357672169344966e-05
                   649            middle east    1.4235223005488745e-17

genre_id : Western, model : TF-IDF
, format : <tag_id, tag_name, weight>

                tag_id                tag_name                weight
                   178                  canada      0.2932757726712511
                  1007          talking animals      0.2159043379324899
                   277            culture clash      0.14502867356540064
                   523                   horses      0.143562987203365
                   327               dreamworks      0.10044061924617971
                   310  disney animated feature      0.09566904433935668
                   309                  disney     0.006054988320263475
                   509                 history    6.357672169344966e-05
                   649            middle east    1.4235223005488745e-17
```

Input with command

- Task 1:
  Sample input:

  python src/phase1.py print_actor_vector 1484 TF-IDF

  Sample output:

```
(phase1) D:\asu\asu\CS\515\project\phase1>python src\phase1.py print_actor_vector 1484 TF-IDF
actor_id : 1484, model : TF-IDF, format : <tag_id, tag_name, weight>
              tag_id,             tag_name,              weight
                 998,             survival, 0.41677855711711914
                 673,    mountain climbing, 0.27442745892898673
                 852,           ridiculous, 0.10898882810014357
                 883,               scenic, 0.09917557445844806
                 312,           disturbing, 0.09068321755185776
                1059,               unique, 0.00994636384344459
                1047,           true story,5.701721125108135e-18
```

- Task 2:
  Sample input:

  python src/phase1.py print_genre_vector Western TF-IDF

  Sample output:

```
genre_id : Western, model : TF-IDF
, format : <tag_id, tag_name, weight>

                    tag_id                 tag_name                    weight
                       178                   canada       0.2932757726712511
                      1007          talking animals       0.2159043379324899
                       277            culture clash      0.14502867356540064
                       523                   horses       0.143562987203365
                       327               dreamworks      0.10044061924617971
                       310  disney animated feature      0.09566904433935668
                       309                   disney     0.006054988320263475
                       509                  history    6.357672169344966e-05
                       649             middle east   1.4235223005488745e-17
```

- Task 3:
  Sample input:

  python src/phase1.py print_user_vector 146 TF-IDF

  Sample output:

user_id : 146, model : TF-IDF
, format : <tag_id, tag_name, weight>

tag_id	tag_name	weight
305	directorial debut	0.13596907952152928
107	based on a book	0.08227325676595466
507	hip hop	0.0534378197443812
901	sequel	0.04961258743886336
1047	true story	0.03876007678123345
141	black comedy	0.036715483693957704
132	big budget	0.03131088574702739
537	immigrants	0.02918434941694308
728	nudity (full frontal)	0.027090494345926024
770	parody	0.02438428290627575
227	college	0.02180636045559579
134	biography	0.020358889587471694
645	mental illness	0.02030507851226238
556	interracial romance	0.015536677206009418
1039	trains	0.015508931532578176
66	antarctica	0.015472910940792848
109	based on a play	0.01509378735162514
566	irreverent	0.01501752926141026
1007	talking animals	0.01462220084954583
354	ensemble cast	0.014621964698002981
572	italy	0.014517974280596516
883	scenic	0.013883508952508873
140	black and white	0.01347618605745309
178	canada	0.013297880294469759
855	road trip	0.013011358873773964
959	sports	0.012535852030618553
705	new york city	0.012438226396373032
623	magic	0.011758364539149673
877	satire	0.011604013109141696
847	religion	0.011527539029179445
1050	twist ending	0.011519987133747047
849	remake	0.010755190986291586
995	surreal	0.010156811711749628
484	hackers	0.00784396498956747
801	pornography	0.007837987523207591
248	con men	0.007837377958819525
688	mutants	0.007827801222845253
568	island	0.007774178432215757
161	brazil	0.007770519801403983
914	shark	0.007768429464591888
888	science	0.007307650836697038
197	cheerleading	0.007304219856090158
911	sexy	0.007112592576295499
111	based on a tv show	0.007003352657011244
562	ireland	0.00699559009724789
406	franchise	0.00699353381985657
983	submarine	0.006944361598754724
813	prison	0.006941761405892243
842	reality tv	0.006783453034404812
942	space	0.006658290476142721
861	robots	0.006552467221120896
582	journalism	0.0064703347111757055
1075	video game adaptation	0.0063019699984799065
989	superhero	0.0059536774520201705
331	drugs	0.005835925808866037
503	high school	0.005782932329422481
210	christmas	0.0066635148619545549
159	boxing	0.00016857075783554166
903	serial killer	9.55641170776504e-10

- Task 4:
  Sample input:

  python src/phase1.py differentiate_genre Western IMAX P-DIFF2

  Sample output:

```
genre_diff_id : Western, model : P-DIFF2, format : <tag_id, tag_name, weight>

                tag_id                 tag_name                   weight
                   327               dreamworks       1.1306281828219749
                   178                   canada       1.1306281828219749
                   277             culture clash       1.1306281828219749
                   649               middle east       1.1306281828219749
                   523                   horses       1.1306281828219749
                   309                   disney       1.1306281828219749
                   310  disney animated feature       1.1306281828219749
                  1007           talking animals       1.1306281828219749
                   509                   history       0.8927220646312689
```

IV. Installation and execution instructions

This project is under Python 3.6 with dependencies pandas and numpy. 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. All commands should be in the format given in the project description. The code is tested on windows 10 and Ubuntu 14.

V. 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.

VI. 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.

VII. Appendix

This project is done by group members individually.