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 will build vector space on the given data of MovieLens and IMDB. There are 4 tasks in phase1. For the first three tasks, a <tag, weight> vector is built given field id and model to measure how good is the tags for the given field as features. For the last task, a differentiating vector is built to show the difference between genre1 and genre2.

### Keywords

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

# Introduction

## **Terminology**

TF: Term frequency, is used to measure the frequency a specific term appearing in a document.

TFIDF： TFIDF is the multiply TF and IDF. IDF is inverse document frequency, which measures how much the term can discriminate the document from others.

TF-IDF-DIFF

P-DIFF1

P-DIFF2

## 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

Task1:

Assume all the actors in the given database forms whole documents, whether the acotor’s movie has been given a tag or not. In this case, there will be some actors who do not have any tags.

Task2:

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.

Task4:

# 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.

(0)

,where , means the index of the unique tag whose tagid is j, is the index of the unique tag in the sorted list.

Here is the reason why is the same as (term frequency) by definition,

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

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

,where X is the whole list,

Then compute IDF for term t.

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

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 instead the number of unique genres all the movies have

P-DIFF1 and P-DIFF2

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 . In P-DIFF1, denotes the number of movies in genre, g1, containing tag tj, denotes the number of movies in genre, g1 or g2, containing tag tj. While in P-DIFF1, denotes the number of movies in genre, g1, not containing tag tj, 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.

# **Interface specifications**

# Installation and execution instructions

# Related work

# Conclusions

# Bibliography

A list of publications relevant to the work

 Each publication must be properly cited in the body of the

report

 Each publication should have

• A list of the authors

• Title of the publication

• Name of the conference/journal

• Page numbers

• Date of publication

Appendix Specific roles of the group member.