CSE 515 Multimedia and Web Databases
Project 1 Phase #1

Group Members:
Kushal Bhatt
Arun Srivatsa
Suraj Shah
Ninad Jadhav
Pranav Deshpande
Achyutha Bharadwaj

Abstract

The phase 1 of the course project focuses on vector models and information retrieval using TF-IDF (Term Frequency – inverse document frequency). TF-IDF is useful for numerical analysis of text based data to reflect how important a word is for a document in a collection of documents. The importance of a word increases proportionally with number of times it appears in the document but it is offset by the frequency of the word in the corpus. In this project we are counting TF-IDF for actors, genres and users on a given imdb+MovieLens dataset. The whole phase is divided into 4 major subtasks. Task 1 focuses on finding weighted tag vector for a given actor, Task 2 is for genres and Task 3 on users. Task 3 output can help us in user modeling. Task 4 tries differentiates two genres by using TF-IDF difference or probabilistic relevance feedback models.

Keywords
TF, IDF, TF-IDF, Probability relevance feedback

Assumptions

- This implementation is based on the given dataset structure and schema. It will work only if the data is provided in a similar structure.
- For TASK 3 in computing tag-vector for a user it is assumed that mlratings table describes user's watched movie history – a user u has watched a movie x if there an entry for u with movie x. Since mlratings has huge data the model is working as intended.
- In IDF computation the base of log function is assumed to be 10.
- Any other specific assumptions are mentioned for each task separately.
- Task1,2,4 requires joined data over 2 or more tables. In order for faster consecutive executions some helper data tables for tasks are precomputed.
  In the implementation this additional tables will be created by performing join if they do not exist.
  Hence the very first execution can take longer. But after that it will be faster.
- Relative tag weights for newer/older tags are considered for TF computation only. So while computing TF-IDF they will get included from TF values. it is specific to this implementation. IDF is just for rarity of a feature.

Description of Implementation

For tasks 1,2,3 while computing tag vectors newer tags are required to be given more weights because they are considered more descriptive for a movie.
In order to give more importance to newer tags and actor ranks , normalization is performed over data. The tag timestamps are first converted to miliseconds precision.For this UNIX Epoch time used as a reference t0. No of miliseconds from Epoch to a given timestamp is calculated and then weights are computed.
Now, based on latest and oldest timestamp corresponding weight values in the range 0.5 to 1 is assigned. Meaning that the oldest timestamp will get value 0.5 and The latest will get 1.0 .

e.g. for a timestamp t relative weight is computed as

weight = (t - min) / (max-min)

This value will be in [0,1] and then scaled to [0.5,1] using formula $y = ax+b$
0.5 is selected as a base value so that the tag doesn't loose it important at all. Otherwise during tf computation the older tags will get near 0 values and can loose importance by larger factor.

0 should be scaled to 0.5 and 1 to 1.

$0.5 = a(0)+b$ and $1 = a(1)+b$ solving the equation gives $a = b = 0.5$

hence relative weight = 0.5*(weight) +0.5

This implementation is dynamic and computes weights based on currently provided latest and oldest timestamps. Also, it is assumed that any tag with even a second later timestamp is more important.

TF-IDF is computed from TF and IDF vlaues for all tasks.

TF-IDF = TF * IDF

Output of all tasks is descending order of tag weight scores.

Task-1:

In this task the goal is to find a weighted tag vector for a given actor by considering all the movies in which he/she appears. Here the corpus is taken as all the actors since we are interested in them. One document is one actor. Consider all the movies played by this actor and get tags for those movies to find tf , idf.

TF for an actor for tag t = (no of times tag t appears for actor) / (no of tags)

But it is required that more weight is assigned to movie tags which are more recent and movies in which the actor appears with more important role (lower rank)

In order to give actor ranks relative weights, similar approach to timestamps is used. Here lower the rank, more important it is.

weight = (rank – min_rank) / (max_rank – min_rank)

This weight will be in range 0-1 with highest rank getting weight 1. But we need it the other way round. So using y = ax+b we will map range [0,1] to [0.5,1] with 1 getting relative weight 0.5 and 0 getting 1. (Lowest rank gets highest weight)

relative_weight = -0.5 * (weight)+1

By taking into account relative timestamp weight and actor_rank weight overall relatiive weight is computed by first adding this two weights and then mapping the sum range [1,2] to [0.5,1]. This is just selected for a simplicity. Range could be anything. The value of TF weights might differ but the ranking of TF-weights will not.
We will use this accumulated relative weight in below TF computation.

Now for a given tag t TF = (Sum of weights for all occurrences of t for this actor)

(sum of all weights for this actor)

Here, if a tag was appearing for 5 times then in above revised TF formula numerator will become sum of 5 corresponding relative weights.
And since our corpus is actors, IDF is computed as

IDF = log (no of actors / no of all actors having t).

In order for computing above TF and IDF data from multiple tables are required. For faster computation a separate joined table is pre-computed and named master in the implementaion. It is join over mlmovies, mltags , movie_actor . Mltags has taginfo, mlmovies has actor info and movie_actor gives actor ranks .

TASK:2

Task 2 gives tag vector for a given genre. The TF computation is similar to TASK-1 only that actor movie ranks are not needed. Here the corpus is all the genres since we are interested in them. One document is one genre. Consider all the movies with this genre and get tags for those movies to find tf , idf.
Get all the tags occuring for movies in this genre.

Now for a tag t TF = (Sum of tag_weights for all occurances of t in this genre)

(sum of all tag_weights for this genre)

IDF = log(no of genres / no of all genres having t)

Task 2 also requires data from multiple tables. For faster computation a separate joined table is pre-computed and named task2 in the implementation. It is a join over mlmovies and mltags.

TASK 3:

Task 3 gives tag vector for a given userid. The TF computation is similar to TASK-2. Here the corpus is all the users since we are interested in user behaviors. Output of this task can be used to gather information about use trends and can be used for user modeling. One document is one user. Consider all the movies watched by this user and get tags for those movies to find tf , idf.
First get all the tags for the movies watched by this user from mlratings.

Now for a tag t TF = (Sum of tag_weights for all occurances of t)

(sum of all tag_weights for this user)

IDF = log(no of users / no of all users having tag t in movies watched)

This task 3 requires information from mlratings and mltags. Mlratings gives movies for a particular user and tags needs to be extracted for those movies from mltags.

Due to huge size of mlratings data the IDF computation is taking considerable time.

TASK 4:

The main objective of this task is to produce a differentiating tag vector for given two genres. For computing the differentiating tag vector following three models are used:

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

TF-IDF-DIFF:

For this model tfs are computed same as task2 for both genre. For IDF instead of considering all movies only movies in genre1 and genre2 are considered.

Here for IDF calculation genre,movie tuples are selected from the dataset. We are considering all tags present in set of movies with genre1 or genre 2.

If a tag is not present for genre then tf =0 hence TF-IDF will be 0.

For genre g and tag t:

IDF = log (no of movies with g1 or g2 / no of movies in g having t)

TF-IDF is computed for both genres. The differentiating tag vector is computed by subtracting TF-IDF values for each tag. Output is printed in descending order of resulting differences.

P-DIFF1:

The objective of this task is to compute the tag vector which is used to discriminate between the given genres g1 and g2 using given formula.

$W_{1,j} = \log((r_{1,j} / R - r_{1,j}) / ((m_{1,j} - r_{1,j}) / (M - m_{1,j} - R - r_{1,j}))) * (r_{1,j} / R - ((m_{1,j} - r_{1,j})/ (M-R)))$

For a special cases where r = 0 or r =m log is undefined. For such cases alternative formula mentioned in the book[2] is used. Smoothing values 0.5 for numerator and 1 for denominator are added in the probability calculation.

$p(f_k |relevant) = (r_k + 0.5) / (|R| + 1)$ let's call this a .

and

$p(f_k |irrelevant) = ((d_k - r_k) + 0.5) / (|D - R| + 1)$ let's call this b

$W_{1,j} = \log (a\ (1 - b) / b\ (1 - a)\ ) * (r_{1,j} / R\ -\ ((m_{1,j} - r_{1,j})/ (M-R))\ )$

where,

$r_{1,j}$ 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 = |movies(g1)|

M = |movies(g1) | ∪ |movies(g2)|

This values are computed for both genres. And the differentiating tag vector is computed by subtracting TF-IDF values for each tag. Output is printed in descending order of resulting differences.

P-DIFF2

The goal is similar to P-DIFF1 but the method of differentiating is slightly different.

The formula structure is similar to P-DIFF1 with but the meaning of terms changes.

$W_{1,j} = \log(\ (r_{1,j} / R - r_{1,j}) / (\ (m_{1,j} - r_{1,j}) / (M - m_{1,j}\ - R - r_{1,j}))\ \ ) * (r_{1,j} / R\ -\ ((m_{1,j} - r_{1,j})/ (M-R))\ )$

For a special cases where r = 0 or r =m log is undefined. For such cases alternative formula mentioned in the book[2] is used. Smoothing values 0.5 for numerator and 1 for denominator are added in the probablity calculation.

$p(\ f_k\ |\text{relevant}) = (r_k + 0.5) /\ (|R| + 1)$ let's call this a .

and

$p(\ f_k\ |\text{irrelevant}) = ((d_k - r_k\ ) + 0.5) / (|D - R| + 1)$ let's call this b

$W_{1,j} = \log (a\ (1 - b) / b\ (1 - a)\ ) * (r_{1,j} / R\ -\ ((m_{1,j} - r_{1,j})/ (M-R))\ )$

where,

$r_{1,j}$ is the number of movies in genre, g2 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 = |movies(g1)|

M = |movies(g1) | ∪ |movies(g2)|

System requirements/installation

Solution was implemented on Ubuntu with Python 2.7 and MySQL.

It is tested on Ubuntu 16.04 , but should work with any installation.

Python 2.7 is must. There will be errors if executed on python 3 due to syntax differences.

MySQL installation is required. If not installed please install it with following command:

```
sudo apt-get update

sudo apt-get install mysql-server
```

Follow prompts to configure root user and it should be running.

Execution Insructions/ User Interface

Given program contains the file core.py which has all logic and computation. The first few lines has the mysql connector intialization that requires Database name and mysql credentials.

It is assumed that the whole dataset (all tables) is under a single mysql db. Please put your userid and password of mysql to execute the program.

For more details please check provided README file.

Please move to the folder that contains code.

Value of model for TASK 1,2,3 can be “TF” or “TF-IDF”.

TASK 1: python print_actor_vector.py actorid model

e.g. python print_actor_vector.py 123 TF-IDF

TASK 2: python print_genre_vector.py genre model

e.g. python print_actor_vector.py Action TF

TASK 3: python print_actor_vector.py userid model

e.g. python print_user_vector.py 1988 TF-IDF

TASK 4: python differentiate_genre.py genre1 genre2 model

model can be TF-IDF-DIFF, PDIFF-1 or P-DIFF2 .

e.g. python differentiate_genre.py Action Comedy P-DIFF1

If you face any other troubles while execution feel free to contact me on: Kushal.Bhatt@asu.edu (#) 602-702-4309

Conclusion

Bibliography

[1] https://en.wikipedia.org/wiki/Tf%E2%80%93idf

[2] K. Selcuk Candan, Maria Sapino, “Data Management for Multimedia Retrieval”,2010, pages 404-410

[3] https://www.coursera.org/learn/ml-foundations/lecture/1rg5n/calculating-tf-idf-vectors