Experiment and Analysis of Imdb movie dataset using vector models like TF-IDF

Group Structure and Members

Group 3	Member 1	Deshpande	Pranav
	Member 2	Jadhav	Ninad
	Member 3	Bharadwaj	Achyutha Sreenivasa
	Member 4	Ramesh	Arun Srivatsa
	Member 5	**Shah**	**Suraj**
	Member 6	Bhatt	Kushal

Abstract

In this experiment, we examine the results of applying term frequency (tf) and tf - inverse document frequency (idf) on imdb movie dataset consisting of several factors like actors, users, tags, movies. In the last task, we also examine 3 different models viz. tf-idf-diff, pdiff1 and pdiff2 for genre differentiation. Unlike the conventional count to be considered for frequency, we use a normalized timestamp value of the tag, thus giving more importance (and a higher value) to the recent tags. Similarly, we consider normalized actor ranks while defining tag weights for actors. In information retrieval, we might want to query a word (which we associate with a tag here) and then show relevant documents based on that keyword. To define the order of this relevancy, we have models like tf-idf for evaluation in this experiment. Instead of tf, we use relative tf so that we have bounds for the resultant weights. tf-idf calculates values for a word in a document through an inverse proportion of tf to the fraction of documents the word appears in. The idf part helps weigh down frequent words like prepositions (of, by, at, on) in a document thus helping us shape the logic that highly frequent words carry less information while less frequent words carry more.

Keywords

term frequency (tf), inverse document frequency (idf), vector space models, information retrieval

Introduction

We divide this experiment into 4 subtasks, each of whose specifications and assumptions are mentioned in the coming sections.

- **Terminology**

  We consider the movielens+imdb database which has the following schema:

  – mlmovies(movieId, movieName, genres)
  – mltags(userId,movieId,tagid,timestamp)
  – mlratings(movieId,userId,imdbId,rating,timestamp)
  – genome-tags(tagId, tag)
  – movie-actor (movieId, actorId, actorMovieRank)
  – imdb-actor-info (actorId, name, gender)
  – mlusers(userId)

  We want to define a relationship between actor-tag, user-tag and genre-tag which will basically be our first 3 subtasks. In the last task, we will use 3 different models to find a tag vector that describes the difference between 2 genres.

  While visualizing our schema in terms on documents and words, for the first subtask, we can relate actor to be the document and the words to be tags. The tricky part here is relating the actor to the tag, which happens via movies. This extra layer of movies can be integrated into the actor and we visualize each distinct document as (actor, movie) pair. Similarly for the second task, (genre,movie) becomes a document and tag is the word and for the third task (user,movie) becomes a document. For the final task, we use simple Manhattan distance for calculating the difference using tf-idf-diff model. The custom formulas used for pdiff1 and pdiff2 models are mentioned below in the implementation section.

- **Goal Description/Problem Specification**

  In information retrieval, tf-idf is useful to find relevant documents associated with keywords in a query. Similarly, for our imdb dataset, the problem specification is to construct a graph that relates the keywords (that we call tags) to different components pertaining to the movie world like genres, actors, users, etc. The purpose of this experiment is to finally build a model that can be used to query relevant information about any of the above components just using the tags. For example, if a user searches for keyword “Drama”, the movie section will show drama movies, the actor section will show actors who have acted in drama movies and the users section will show all users who’ve watched drama movies. Our task here is building the model using some known techniques and compare them to analyze the results.

- **Assumptions**

Task 1: Actor-Tag relation

1. For the tf model in task 1, we use normalized timestamp weights instead of simple count associated with each tag to determine its contribution to the tf. We also inversely multiply it by the normalized actor's rank (lower value has higher importance). We use relative tf to define the tag weight range between 0 and 1.
2. For the tf-idf model, while calculating we consider the total documents as a combination of the <actor,movie> pair. The numerator thus becomes the count of such unique pairs of <actor,movie>. The denominator for idf computation is summation of the normalized timestamps for each tag.

Task 2: Genre-Tag relation

1. For the tf model in task 2, we use normalized timestamps from mltags table just like task 1 but actor ranks are not relevant here. We consider all movies in a genre and then find all tags associated with these movies to weight them using tf model. We use relative tf to define the tag weight range between 0 and 1.
2. For the tf-idf model, while calculating we consider the total documents as a combination of the <genre,movie> pair. The numerator thus becomes the count of such unique pairs of <genre,movie>. The denominator for idf computation is summation of the normalized timestamps for each tag.

Task 3:User-Tag relation

1. For the tf model in task 3, we use normalized timestamps like task 2. We assume that all users who have tagged a movie have also watched the movie. Thus, since tag information was needed, no users' data from mlratings table is made use of. For a user, we find all the movies he has tagged and all the tags associated with these movies are then considered to weight them using tf model. We use relative tf to define the tag weight range between 0 and 1.
2. For the tf-idf model, while calculating we consider the total documents as a combination of the <user,movie> pair. The numerator thus becomes the count of such unique pairs of <user,movie>. The denominator for idf computation again is summation of the normalized timestamps for each tag under consideration.

Task 4:tf-idf-diff, p-diff1, p-diff2 on two genres

1. For the tf-idf-diff model, we simply calculate tf-idf vectors for both the genres and then use Manhattan distance (without the absolute values) on the final tag vector to calculate the difference. We keep the original signs with the assumption that diff(genre1, genre2) != diff(genre2, genre1).
2. For p-diff1 and p-diff2 models, we use direct formulae to calculate tag weights, while handing edge cases like divide by zero and negative values for logarithm.

Miscellaneous Assumptions:

1. All log computations are done to base 10
2. While normalizing timestamp (whose values are in seconds standardized from year 1970) and actor rank, max = max+1 and min = min-1 than the actual dataset values to take care of corner cases and prevent multiplication by weight 0 for min values.
3. Lower numerical actor rank in dataset relates to higher importance. Hence the normalized rank between 0 to 1 is taken as (1 – normalized rank) to reverse the relation and assign actor with lower rank a higher normalized weight value.
4. For pdiff1 and pdiff2 models, if m1j =r1j, then term becomes undefined(log(0)). Hence, the tag is assigned a 0 weight here directly meaning both these tags had a similar score in their respective genres, bringing the difference to 0 in the diff model.
5. If a tag is not found, it is initialized with a 0 weight across all tasks.
6. MLTags table is used for all normalized timestamp weights, MLRatings is not used anywhere. This is done on purpose to compare results with other teammates some of whom consider both tables and some consider only MLRatings. This mainly comes into play for task 3.

Proposed Solution/Implementation

Task 0:Preprocessing/Meta-data generation

Preprocessing of the dataset involves model creation for each of the tables with proper foreign key relations to clean and store any data inconsistencies. While putting all the data into the MySQL database, we convert the date from "YYYY-MM-DD HH-MM-SS" format into seconds from the UNIX Epoch time. After converting it to milliseconds, we make another meta-data column for both mltags and mlratings and using the standard normalization on this data to convert it into values between range of 0 and 1. Same thing is done for movie_actor_rank column in the movieactor table. The only difference in storing the final actor rank weight is that the weight is stored as 1 – norm_weight_value. The reason for doing this is that numerical actor rank value in the given database is inversely proportional to its importance – lower rank needs higher weights. Also, to handle the edge cases, instead of taking the database max and min values for normalization we take max = max+1 and min=min-1 to avoid having weight = 1 for the latest tag and min = 0 for the lowest tag, which would mean it will be ignored from the computations.

We also have 4 more computations tables specific to each task that help speed up execution of the search queries. Each of these tables can be populated by running the tf() method in every task, in the implementation scripts. These methods need to run only once.

Please note that for idf in each task the idf value is normalized between 0 to 1 by finding the range of idf which will be min=0 and max=log(N). It was noted that normalizing the idf does not

change the results in any way since the ratio between each normalized idf tag value is still constant.

Task 1: Actor-Tag relation

In the first task, we define actor-movie-tag relationship to give out a <tag,weight> sorted vector as output for an input actor and tf/tf-idf model.

The **relative term frequency** for an actor is calculated as:

$$\text{tf(actor,t)} = \frac{c(t)}{C(T)}$$

where,

c(t) = sum(normalized_weights * normalized_actor_movie_rank) for tag t

C(T) = sum(normalized_weights * normalized_actor_movie_rank) for all tags T in all movies the actor has acted in. (All other tag values will be 0)

The **inverse document frequency** for an actor is calculated as:

$$\text{idf(actor,t)} = \log\left(\frac{N}{n(t)}\right)$$

where,

N = Total number of unique (actor,movie) pairs visualized as a document

n(t) = sum(normalized_weights) for tag t under consideration

The **tf-idf** is simply calculated as below:

$$\text{tf-idf(actor,t)} = \text{tf(actor,t)} * \text{idf(actor,t)}$$

Task 2: Genre-Tag relation

In the second task, we define genre-movie-tag relationship to give out a <tag,weight> sorted vector as output for an input genre and tf/tf-idf model.

The **relative term frequency** for a genre is calculated as:

$$\text{tf(genre,t)} = \frac{c(t)}{C(T)}$$

where,

c(t) = sum(normalized_weights) for tag t

C(T) = sum(normalized_weights) for all tags T in all movies corresponding to the given genre. (All other tag values will be 0)

The **inverse document frequency** for a genre is calculated as:

$$\text{idf(genre,t)} = \log\left(\frac{N}{n(t)}\right)$$

where,

N = Count of unique (genre,movie) pairs visualized as a document

n(t) = sum(normalized_weights) for tag t under consideration

The **tf-idf** is simply calculated as below:

tf-idf(genre,t) = tf(genre,t)*idf(genre,t)

Task 3:User-Tag relation

In the third task, we define user-movie-tag relationship to give out a <tag,weight> sorted vector as output for an input user and tf/tf-idf model.
The **relative term frequency** for a user is calculated as:

$$\text{tf(user,t)} = \frac{c(t)}{C(T)}$$

where,
c(t) = sum(normalized_weights) for tag t
C(T) = sum(normalized_weights) for all tags T in all movies corresponding tagged by the user. (All other tag values will be 0)

The **inverse document frequency** for a user is calculated as:

$$\text{idf(user,t)} = \log\left(\frac{N}{n(t)}\right)$$

where,
N = Count of unique (user,movie) pairs visualized as a document
n(t) = sum(normalized_weights) for tag t from only MLTags table

The **tf-idf** is simply calculated as below:

tf-idf(user,t) = tf(user,t)*idf(user,t)

Task 4:tf-idf-diff, p-diff1, p-diff2 on two genres

The **tf-idf-diff** for two genre involves tag vector calculation for both genre using the exact task 2 method for **tf-idf** calculation. After that, the diff between them is calculated as below:

tf-idf-diff(genre1,genre2,t) = tf-idf(genre1,t) - tf-idf(genre2,t) …for all t in T

where,
T = set of all tags
This also means that tf-idf-diff(genre1,genre2,t) != tf-idf-diff(genre2,genre1,t) since we do not consider absolute values like in Manhattan distance.

The **p-diff1** between two genres is calculated as:

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

where,
r1,j is the number of movies in genre, g1, containing the tag tj
m1,j is the number of movies in genre, g1 or g2, containing the tag tj
R = count(movies(g1)), and
M = count (movies(g1) ∪ movies(g2))

The **p-diff2** between two genres is calculated as:

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

where,
r1,j is the number of movies in genre, g2, not containing the tag tj
m1,j is the number of movies in genres, g1 or g2, not containing the tag tj
R = count(movies(g2)), and
M = count(movies(g1) ∪ movies(g2)).

User Interface Specifications

Task	Call / Input	Parameter type/values	Output	Goal
1	python print_actor_vector.py <actorid> <model>	<actorid> = <integer> <model> = tf or tf-idf	Weighted and sorted tag vector for input actor	Model gives relevant query results for actors associated with a keyword/tag
2	python print_genre_vector.py <genre> <model>	<genre> = <string> <model> = tf or tf-idf	Weighted and sorted tag vector for input genre	Model gives relevant query results for genres associated with a keyword/tag
3	python print_user_vector.py <userid> <model>	<userid> = <integer> <model> = tf or tf-idf	Weighted and sorted tag vector for input user	Model gives relevant query results for users associated with a keyword/tag
4	python differentiate_genre.py <genre1> <genre2> <model>	<genre1> = <genre2> = <string> <model> = tf-idf-diff or p-diff1 or p-diff2	Weighted and sorted tag vector diff for input genres	Model gives relevant query results for comparing two associated with a keyword/tag

System Requirements/Installation and execution instructions

- **System Requirements**
  The system hardware requirements are as follows:
  - HDD: 20 GB

- RAM: >4 GB
- Processor: 2.2 GHz (intel i5 5th generation or better)

The system software requirements are as follows:

- OS: Unix/Linux, preferably Ubuntu 14.04+
- Terminal, file editor

- **Execution Instructions**

Detailed setup instructions are provided in the readme.md file inside code/MWDBProject folder in the submission zip file.

Instructions for executing all 4 tasks:

Task1: To test task 1 run the command: (in the MWDBProject/ mwd_proj/ med_proj/ scripts/ directory)
<actor-id> eg: 1575755, 506840,
<model> eg: TF, TF-IDF
> python print_actor_vector.py <actor-id> <model> #eg: python print_actor_vector.py 506840 TF

Task2: To test task 2 run the command: (in the MWDBProject/ mwd_proj/ med_proj/ scripts/ directory)
<genre> eg: Action, Documentary
<model> eg: TF, TF-IDF
> python print_genre_vector.py <genre> <model> #eg: python print_genre_vector.py Documentary TF-IDF

Task3: To test task 3 run the command: (in the MWDBProject/ mwd_proj/ med_proj/ scripts/ directory)
<userid> eg: 146, 9316, 1988, 30167
<model> eg: TF, TF-IDF
> python print_user_vector.py <userid> <model> #eg: python print_user_vector.py 1988 TF

Task4: To test task 4 run the command: (in the MWDBProject/ mwd_proj/ med_proj/ scripts/ directory)
<genre> eg: Action, Documentary, Horror
<model> eg: TF-IDF-DIFF, P-DIFF1, P-DIFF2
> python differentiate_genre.py <genre1> <genre2> <model> #eg: python differentiate_genre.py Action Drama P-DIFF2

NOTE: Please note that if you run tf() method along with each task, it will take longer to execute. Please comment the call to this method if comment removed

Related work

Conclusions

References

Mention one that Ninad said

Roles of group members