FEATURE SELECTION OF MOVIE LENS AND IMDB DATA
BY
SATHEESH KURUNTHIAH – 1212901792

GROUP MEMBERS

Last Name	First Name
Shah	Mihika
Hashmi	Syed Usama
	Vaishnavi Raj
Dikshit	Ishan
Kurunthiah	Satheesh
Sadaye	Raj

ABSTRACT

Multimedia objects cannot be stored with all available features due to dimensionality curse and cost factor. Initial step for any multimedia database design is feature selection or dimensionality reduction. Vector space model is an algebraic model for representing text documents (and any objects, in general) as vectors of identifiers, such as, for example, index terms. It is used in information filtering, information retrieval, indexing and relevancy rankings. This project aims in obtaining discriminating tag vectors for given inputs in a database so that it can be used for feature selection or dimensionality reduction.

Keywords: term; document; term frequency; inverse document frequency;

INTRODUCTION

TERMINOLOGY

- Term – A term is any feature of an object in a vector space or dataset **[2]**
- Document – A document is an object that can have many features or terms **[2]**
- Term Frequency (TF) – It is the raw count of a term in a document, i.e. the number of times that term t occurs in document d **[4]**
- Inverse Document Frequency (TF-IDF) – It is a measure of how much information the word provides, that is, whether the term is common or rare across all documents D **[4]**
- Pandas – It is a powerful Python data analytics toolkit. It is used to read input files, process and store results as csv file
- Data Frame – It is a 2-dimensional labeled data structure with columns of potentially different types. It is like a spreadsheet or SQL table

GOAL DESCRIPTION

The goal of this project is to learn the steps involved in representing giving dataset in vector format and identifying vectors that has more discriminating weight. The goal of each task is as follows

- The goal of Task 1 is to consider all movies an actor has played and choose tags associated with those movies with more discriminating weights.
    - Gets all movies an actor has played

    - Get all tags associated with those movies
    - Associate weight to tags based on timestamp (newer timestamp will have more weight)
    - Associate weight to actor rank based on actor-movie-rank (actor with lower rank will have more weight)
    - For each tag calculate TF as follows
        - TF = count of a given tag in movie / total tag count for given movie **[2]**
    - For each tag calculate TF IDF by multiplying TF weight with IDF weight as follows
        - IDF = log (total actors count in dataset / total actors count for given tag in entire dataset) **[2]**
    - Print tags in order of high discriminatory weight to low discriminatory weight
- The goal of Task 2 is to consider all movies under a given genre and tags associated with those movies with more discriminating weights.
    - Get all movies under a given Genre
    - Get all tags associated with those movies
    - Associate weight to tags based on timestamp (newer timestamp will have more weight)
    - For each tag calculate TF as follows
        - TF = count of a given tag in genre / total tag count for given genre **[2]**
    - For each tag calculate TF IDF by multiplying TF weight with IDF weight as follows
        - IDF = log (total genre count in dataset / total genres count for given tag in entire dataset) **[2]**
    - Print tags in order of high discriminatory weight to low discriminatory weight
- The goal of Task 3 is to consider all movies watched by user (both review and rating) and tags associated with those movies with more discriminating weights
    - Get all movies watched by given users
    - Get all tags associated with those movies
    - Associate weight to tags based on timestamp (newer timestamp will have more weight)
    - For each tag calculate TF as follows
        - TF = count of a given tag in movie / total tag count for given movie **[2]**
    - For each tag calculate TF IDF by multiplying TF weight with IDF weight as follows
        - IDF = log (total user count in dataset / total users count for given tag in entire dataset) **[2]**
    - Print tags in order of high discriminatory weight to low discriminatory weight
- The goal of Task 4 is to consider given two genres and get more discriminating tags for genre 1 with respect to genre 2
    - Get all movies under given Genre 1 and Genre 2
    - Get all tags associated with those two genres
    - Associate weight to tags based on timestamp (newer timestamp will have more weight)
    - For each tag calculate TF-IDF-DIFF as follows
        - For each tag calculate TF for Genre 1 as follows
        - TF = count of given tag in Genre 1 / Total tag count in Genre1
        - Combine movies in Genre 1 and Genre 2 and calculate IDF
        - For each tag calculate TF-IDF-DIFF as follows
        - TF-IDF-DIFF = total genre count / total genres count for given tag in Genre 1 + Genre 2
    - For each tag calculate P-DIFF1 as follow
        - R = no of movies in Genre 1
        - M = no of movies in Genre 1 + no of movies in Genre 2
        - For each tag in Genre 1 calculate

- r = no of movies in Genre 1 containing this tag
- m = no of movies in Genre 2 containing this tag
- calculate weight for each tag as follows

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

  - Print tags in order of high discriminatory weight to low discriminatory weight
- For each tag calculate P-DIFF2 as follows
  - R = no of movies in Genre 1
  - M = no of movies in Genre 1 + no of movies in Genre 2
  - For each tag in Genre 1 calculate
    - r = no of movies in Genre 2 not containing this tag
    - m = no of movies in Genre 1 not containing this tag
    - calculate weight for each tag as follows

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

  - Print tags in order of high discriminatory weight to low discriminatory weight

ASSUMPTIONS

- The files in Input directory is hard coded in the program. Any change in file name will not work
- The column names in each input files are also hard coded. Any change in column names will not work
- Task 1, Task 2 and Task 3 takes in 3 parameters <vector-model> <input> <model>
  - Task 1
    - vector-model: print_actor_vector
    - input: Any valid actor id in dataset
    - model: TF or TF-IDF
  - Task 2
    - vector-model: print_genre_vector
    - input: Any valid genre name in dataset
    - model: TF or TF-IDF
  - Task 1
    - vector-model: print_user_vector
    - input: Any valid user id in dataset
    - model: TF or TF-IDF
  -
- Task 4 takes in 4 parameters <vector-model> <genre1> <genre2> <model>
  - vector-model: differentiate_genre
  - genre1: Any valid genre name in dataset
  - genre2: Any valid genre name in dataset
  - model: TF-IDF-DIFF or P-DIFF1 or P-DIFF2

DESCRIPTION OF THE IMPLEMENTATION

The implementation of all tasks is done using Python. Tags which has divide by zero edge case are omitted on all tasks After reading input file we set weight for timestamp and actor rank as follows

ML TAGS TABLE

- Get oldest timestamp from this table
- For each row get delta value (row.timestamp – oldest timestamp)
- This delta will give days and seconds for each row
- Final Weight = (delta.days * 24 * 60 * 60) + delta.seconds
- Normalize this weight in range 1 – 2. i.e., {A, B} → {a, b} using the formula ((weight - A) * (b - a)) / (B - A) + a **[5]**
  where A = 1, B = latest_timestamp delta in seconds, a = 1, b = 2

MOVIE ACTOR TABLE

- Get highest rank in this table
- For each rank in the table weight = highest rank / row.rank
- Normalize this weight in range 1 – 2. i.e., {A, B} → {a, b} using the formula ((weight - A) * (b - a)) / (B - A) + a **[5]**
  where A = 1, B = latest_timestamp delta in seconds, a = 1, b = 2

Task 1 (Document: ACTOR, Feature: Tag)

- Input
  - Consider actor id as 67729
  - Consider model as TF or IDF
  - Open command prompt and type “python MwdbPhase1.py print_actor_vector 67729 tf-idf”
- Implementation
  - Combine ml_tags, genome_tag, movie_actors and ml_movie to a single data frame.
  - Iterate through movies done by actor id and calculate TF value (multiply count of tag with normalized timestamp weight and normalized actor weight) for each tag and store this is a dictionary to have unique tag as key and their weight as values.
  - If a tag repeats for same movie or across movies we add their weight and have single entry in dictionary
  - Now this dictionary has all unique <Tag, TF Weight> pairs. Iterate through this dictionary and calculate TF IDF value as TF Weight * log (Total actors count / total actors count associated with given Tag)
  - Sort this dictionary based on values (i.e., TF-IDF weight) in descending order
  - Print the result as sequence of <tag, weight> pairs
- Output
  - These tags better discriminate the given user

```
<cannibalism, 0.0264307433584>
<psychology, 0.0257234550177>
<serial killer, 0.0253598435913>
<time travel, 0.0182319803416>
<atmospheric, 0.0134541625817>
<sci-fi, 0.0107111205454>
<boring, 0.0101494510462>
<hannibal lecter, 0.00999345551583>
<nudity (rear), 0.00549502345595>
<action, 0.00502096236974>
<robots, 0.00339170833447>
<heroine in tight suit, 0.00309039426887>
<alternate universe, 0.00242477281965>
<nuclear, 0.00213524677146>
<fun, 0.00213037004021>
```

```
cannibalism, 0.226394597449
serial killer, 0.170526584653
psychology, 0.147320143699
time travel, 0.118542211453
atmospheric, 0.0874130181681
hannibal lecter, 0.0807200548473
sci-fi, 0.0598297060539
boring, 0.0507553090033
nudity (rear), 0.0352675579744
robots, 0.0254279711239
heroine in tight suit, 0.0242308091467
action, 0.0233776929503
alternate universe, 0.0231943538703
nuclear, 0.0204248698337
fun, 0.0147051201987
```

Task 2 (Document: GENRE, Feature: Tag)

- Input
  - Consider Genre name as Mystery
  - Consider model as TF or IDF
  - Open command prompt, type “python MwdbPhase1.py print_genre_vector Mystery tf-idf”

- Implementation
  - Combine ml_tags, ml_movies and genome_tags
  - Iterate through movies under given genre and calculate TF value (multiply count of tag with normalized timestamp weight and divide by total count of tags in given genre) for each tag and store this is a dictionary to have unique tag as key and their weight as values.
  - If a tag repeats for same movie or across movies we add their weight and have single entry in dictionary
  - Now this dictionary has all unique <Tag, TF Weight> pairs. Iterate through this dictionary and calculate TF IDF value as TF Weight * log (Total genres count / total genres count associated with given Tag)
  - Sort this dictionary based on values (i.e., TF-IDF weight) in descending order
  - Print the result as sequence of <tag, weight> pairs

- Output
  - The above tags better discriminate the Genre “Mystery” for TF and TF-IDF models

```
time travel, 1.198807
philip k. dick, 0.497734
serial killer, 0.354694
psychology, 0.324971
drama, 0.295999
surreal, 0.275094
future, 0.214495
mystery, 0.213275
based on a book, 0.200209
hannibal lecter, 0.198572
nudity (full frontal - notable), 0.192533
twist ending, 0.179333
mindfuck, 0.177105
sci-fi, 0.153142
remake, 0.143704
surprise ending, 0.13741
disturbing, 0.13582
```

```
philip k. dick, 0.9586354
time travel, 0.94525439
serial killer, 0.58984371
hannibal lecter, 0.44637546
psychology, 0.35031955
nudity (full frontal - notable), 0.32017564
mystery, 0.30723812
drama, 0.2740949
psychological, 0.27048268
surprise ending, 0.26465158
oscar (best actor), 0.25810254
mindfuck, 0.25513261
death, 0.24699533
future, 0.23122615
life & death, 0.22883657
science, 0.21835019
```

Task 3 (Document: USER, Feature: Tag)

- Input
  - Consider User id as 144
  - Consider model as TF or IDF
  - Open command prompt, type “python MwdbPhase1.py print_user_vector 144 tf-idf”

- Implementation
  - Combine ml_movie, ml_tags and genome_tags
  - Iterate through movies watched by user (get users from ml_ratings table and ml_tags table) and calculate TF value (multiply count of tag with normalized timestamp weight and divide by total count of tags in given movie) for each tag and store this is a dictionary to have unique tag as key and their weight as values.

- If a tag repeats for same movie or across movies we add their weight and have single entry in dictionary
- Now this dictionary has all unique <Tag, TF Weight> pairs. Iterate through this dictionary and calculate TF IDF value as TF Weight * log (Total users count / total users count associated with given Tag)
- Sort this dictionary based on values (i.e., TF-IDF weight) in descending order
- Print the result as sequence of <tag, weight> pairs

- Output
  - These tags better discriminate the user-id "144" by TF and TF-IDF models

```
hilarious, 1.35278222752
nudity (topless), 1.04166072428
based on a book, 1.00947842945
cynical, 1.0
rock and roll, 1.0
disturbing, 0.615170458279
crime, 0.506584960331
werewolves, 0.5
not funny, 0.421118198135
movielens top pick, 0.297982366138
visceral, 0.281289315385
directorial debut, 0.276323947925
ireland, 0.27446431312
murder, 0.272514363198
```

```
cynical, 8.82017896242
hilarious, 7.43793605603
rock and roll, 6.82017896242
based on a book, 5.5503656505
nudity (topless), 5.45331937128
disturbing, 3.99753196419
werewolves, 3.41008948121
not funny, 3.04688131946
crime, 2.48899799239
visceral, 2.48102210191
movielens top pick, 2.33027543085
directorial debut, 2.16090272437
irish, 1.98926512069
ireland, 1.98580871643
murder, 1.97170040666
gritty, 1.9290673793
nudity (topless - notable), 1.69353631578
mental illness, 1.54297790826
immigrants, 1.24520149788
nudity (topless - brief), 1.18784142921
biographical, 1.14663841725
```

Task 4 (Document: GENRE, Feature: Tag)

- Input
  - Consider two Genre names as Mystery and Comedy
  - Consider model as TF-IDF-DIFF or P-DIFF1 or P-DIFF-2
  - Open command prompt, type "python MwdbPhase1.py differentiate_genre Mystery Comedy tf-idf-diff"

- Implementation
  - Combine ml_tags, ml_movies and genome_tags
  - TF-IDF-DIFF
    - Iterate through each movie in genre 1 and for each tag in those movies calculate TF Values (multiply count of tag with normalized timestamp weight and divide by total count of tags in given genre) and store this is a dictionary to have unique tag as key and their weight as values.
    - If a tag repeats for same movie or across movies we add their weight and have single entry in dictionary
    - Now this dictionary has all unique <Tag, TF Weight> pairs. Iterate through this dictionary and calculate TF IDF value as TF Weight * log (Total genres count in Genre 1 and Genre 2 / total genres count associated with given Tag in union of Genre1 and Genre2)
    - Sort this dictionary based on values (i.e., TF-IDF weight) in descending order
    - Print the result as sequence of <tag, weight> pairs

- P-DIFF1
  - Get tags in Genre 1 and Genre 2 as list and maintain separate copies.
  - Create two dictionaries to get r and m values while calculating P-DIFF1
  - Consider each tag in Genre 1 and if same tag is present in a movie under Genre 1 then increment count in r dictionary for that tag
  - Consider each tag in Genre 1 and if same tag is present in a movie under Genre 2 then increment count in m dictionary for that tag
  - Calculate R by counting number of movies in Genre 1
  - Calculate M by counting number of movies under Genre 1 and Genre2
  - Apply these values in the P-DIFF1 formula to get weight for each tag
  - Sort this dictionary based on values (i.e., TF-IDF weight) in descending order
  - Print the result as sequence of <tag, weight> pairs
- P-DIFF2
  - Get tags in Genre 1 and Genre 2 as list and maintain separate copies.
  - Create two dictionaries to get r and m values while calculating P-DIFF2
  - Consider each tag in Genre 1 and if same tag not is present in a movie under Genre 2 then increment count in r dictionary for that tag
  - Consider each tag in Genre 1 and if same tag is not present in a movie under Genre 1 then increment count in m dictionary for that tag
  - Calculate R by counting number of movies in Genre 1
  - Calculate M by counting number of movies under Genre 1 and Genre2
  - Apply these values in the P-DIFF2 formula to get weight for each tag
  - Sort this dictionary based on values (i.e., TF-IDF weight) in descending order
  - Print the result as sequence of <tag, weight> pairs

- Output
  - These tags discriminate genres Mystery and Comedy based on TF-IDF-DIFF, P-DIFF1 and P-DIFF2 models

<time travel, 0.25023499>
<philip k. dick, 0.16873216>
<serial killer, 0.1022782>
<hannibal lecter, 0.09536528>
<drama, 0.0727831>
<psychology, 0.06777068>
<nudity (full frontal - notable), 0.05553692>
<oscar (best actor), 0.0551451>
<disturbing, 0.05449739>
<future, 0.05271356>
<mystery, 0.05243834>
<twist ending, 0.05172658>
<mindfuck, 0.05103092>
<psychological, 0.04879876>
<surprise ending, 0.04659776>
<death, 0.04459956>

<mystery, 0.0516876770607>
<nudity (full frontal - notable), 0.0514914986583>
<sci-fi, 0.0514914986583>
<serial killer, 0.0456090386246>
<remake, 0.0444284177845>
<time travel, 0.0405132211014>
<disturbing, 0.0340907764721>
<thriller, 0.0299629590334>
<great movie, 0.0299629590334>
<murder, 0.0240078597001>
<action, 0.0240078597001>
<mindfuck, 0.0226501078367>
<sad but good, 0.0226501078367>
<surprise ending, 0.0226501078367>

<detective, 5.18174922871>
<philip k. dick, 5.18174922871>
<oscar (best actor), 5.18174922871>
<alien, 5.18174922871>
<life & death, 5.18174922871>
<death, 5.18174922871>
<cinematography, 5.18174922871>
<paranoid, 5.18174922871>
<treasure, 5.18174922871>
<humanity, 5.18174922871>
<psychiatrist, 5.18174922871>
<exceptional acting, 5.18174922871>
<brutality, 5.18174922871>

SYSTEM REQUIREMENTS

- Windows/ mac OS X/ Linux
- RAM – 4GB or higher to run Python
- Computer should have Python installed in it
- The execution of the project was done on mac OS X El Capitan

RELATED WORKS

The concept for term weighing approach is the work of Gerard Salton and Christopher Buckley, 1988 **[1]**. According to them text indexing systems based on the assignment of appropriately weighted single term produce retrieval results that are superior to those obtainable with other more elaborate text representation. When we are able to represent the document by term vectors, the information request or queries, would be represented either in vector form or in the form of Boolean statements. Also, if we want to measure the similarity of two feature then the similarity value may be obtained by comparing the corresponding vector like a conventional vector product formula. According to them two measures were used to access the ability of a system to retrieve the relevant and reject the irrelevant are recall and precision respectively.

According to Gerard Salton and Christopher Buckley, the differing recall and precision requirements has led to three main considerations. First is a term frequency (tf) factor which will be used as part of term weighing system. Second is a collection dependent factor that favors terms concentrated in a few documents of a collection called inverse document frequency (idf). Third factor is called term discrimination which is obtained by product of term frequency and inverse document frequency.

CONCLUSION

The goal of the project for phase 1 is met. The information extracted from given data set will be used in the following phase to learn different concepts such as feature selection or dimensionality reduction in multimedia data retrieval and query optimization.

BIBLIOGRAPHY

[1] Gerald Salton and Christopher Buckley, "Term Weighing Approaches in Automatic Text Retrievals", Information Processing and Management, 24(5):513–523, 1988.

[2] K. Selcuk Candan and Maria Luisa Sapino, "Data Management for Multimedia Retrieval", Cambridge University Press, UK, 2010.

[3] Spetember 14, 2017, https://pandas.pydata.org/pandas-docs/stable/comparison_with_sql.html

[4] September 14, 2017, https://en.wikipedia.org/wiki/Tf%E2%80%93idf

[5] September 14, 2017, https://en.wikipedia.org/wiki/Normalization_(statistics)