Report

Project Phase 1

Group Members

Hari Kripa Omalur Chandran

Tinu Tomson

Jasmin George

Suprada Ganesh Nadiger

Boasco Paul

Akshay Muraleedharan Nair Santhy

**Abstract**

A program to represent actors, users and genres as weighted tag vectors based on the timestamp of the tags. Also, a program to find the differentiating tag vectors for given two genres.

**Keywords –** Vector model; Tf; Tf-IDF; Pdiff1; Pdiff2; Relevance based feature selection; Imdb dataset; Movie-lens dataset;

**Introduction**

**Terminology:**

* **TF: Term Frequency**, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:   
    
  TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).
* **IDF: Inverse Document Frequency**, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:   
    
  IDF(t) = log\_e(Total number of documents / Number of documents with term t in it).[2]

**Goal Description:**

Using the given sample MovieLens+IMDB data,

1. Implement a program to print the tag vector (as a sequence of <tag, weight> pairs, sorted in descending order of weights) for the given actor under the given vector model. Vector model can be either Tf or Tf-IDF. When combining tag vectors under TF or TF-IDF models, newer tags should be given higher weight than older tags. Similarly, movies where the given actor appears with a lower rank should be given a relatively higher weight.
2. Implement a program to print the tag vector (as a sequence of <tag, weight> pairs, sorted in descending order of weights) for the given genre under the given vector model. Vector model can be either Tf or Tf-IDF. When combining tag vectors under TF or TF-IDF models, newer tags should be given higher weight than older tags.
3. Implement a program to print the tag vector (as a sequence of <tag, weight> pairs, sorted in descending order of weights) for the given user under the given vector model. Vector model can be either Tf or Tf-IDF. When combining tag vectors under TF or TF-IDF models, newer tags should be given higher weight than older tags.
4. Implement a program to print the differentiating tag vector between two given genres using the given model. The models are Tf-IDF- diff, P-Diff1 and P-Diff2.

**Assumptions:**

1. I have applied Salton and Buckley variation given in textbook to derive new formula for task 4.

log(((r1+ (m1/M))/(R-r1 + 1))/(((m1-r1)+(m1/M))/(M- m1 - R + r1 + 1))) \* abs(((r1 + (m1/M))/(R+ 1))-(((m1-r1)+ (m1/M))/(M- count\_g1 + 1)))

[1]

**Description of the Implementation**

I have used pandas to store the data from csv files.

**Problem 1 – Print actors vector:**

* Converted the timestamp column to inverse of difference between the current time and the timestamp.
* Read the command line arguments as actor\_id and model to be used.
* Find all movies the actor has acted and get the tags of all those movies.
* Multiply the timestamp of those tags with the respective actor rank.
* Calculate the total tag weight of all tags by adding all the timestamps
* For each distinct tag in the above found tags,

1. Find all the movies with the tag
2. Count the movies where the given actor has acted
3. Add all the timestamp for the given tag this will be our tag weight
4. For tf model, calculate (tag weight) / (total tag weight)
5. For tf-idf model, calculate ((tag weight) / (total tag weight)) \* log(total\_actors/actors)

**Problem 2 – Print genres vector:**

* Converted the timestamp column to inverse of difference between the current time and the timestamp.
* Read the command line arguments as genre and model to be used.
* Find all movies that has the given genre and get the tags of all those movies.
* Calculate the total tag weight of all tags by adding all the timestamps
* For each distinct tag in the above found tags,

1. Find all the movies with the tag
2. Count the movies where the genre is same as the given genre
3. Add all the timestamp for the given tag this will be our tag weight
4. For tf model, calculate (tag weight) / (total tag weight)
5. For tf-idf model, calculate ((tag weight) / (total tag weight)) \* log(total\_genres/genres)

**Problem 3 – Print users vector:**

* Converted the timestamp column to inverse of difference between the current time and the timestamp.
* Read the command line arguments as user\_id and model to be used.
* Find all movies the user has tagged and rated and get the tags of all those movies.
* Calculate the total tag weight of all tags by adding all the timestamps
* For each distinct tag in the above found tags,

1. Find all the movies with the tag
2. Count the movies where the given user has tagged or rated
3. Add all the timestamp for the given tag this will be our tag weight
4. For tf model, calculate (tag weight) / (total tag weight)
5. For tf-idf model, calculate ((tag weight) / (total tag weight)) \* log(total\_users/users)

**Problem 4 – Print differentiating tag vector:**

* Converted the timestamp column to inverse of difference between the current time and the timestamp.
* Read the command line arguments as actor\_id and model to be used.
* Find all movies the actor has acted and get the tags of all those movies.
* Multiply the timestamp of those tags with the respective actor rank.
* Calculate the total tag weight of all tags by adding all the timestamps
* For each distinct tag in the above found tags,

1. Find all the movies with the tag
2. Count the movies where the given actor has acted
3. Add all the timestamp for the given tag this will be our tag weight
4. For tf model, calculate (tag weight) / (total tag weight)
5. For tf-idf model, calculate ((tag weight) / (total tag weight)) \* log(total\_actors/actors)

**Interface Specifications**

The whole code was written in a Mac OS system.

**System Requirements**

* Python 2
* Pip
* Pandas

**Installation and Execution Instructions**

Install Pip:

Download [get-pip.py](https://bootstrap.pypa.io/get-pip.py).

Run **python get-pip.py**

Install Pandas:

Run **pip install pandas**

Task 1

Run the python file print\_actor\_vector.py with actor\_id and model as arguments.

**python print\_actor\_vector.py actor\_id model**

Running the file creates output\_actor file which contains the tag vector for the given actor and model.

Task 2

Run the python file print\_genre\_vector.py with genre and model as arguments.

**python print\_genre\_vector.py genre model**

Running the file creates output\_genre file which contains the tag vector for the given actor and model.

Task 3

Run the python file print\_user\_vector.py with user\_id and model as arguments.

**python print\_user\_vector.py user\_id model**

Running the file creates output\_user file which contains the tag vector for the given actor and model.

Task 4

Run the python file differentiate\_genre.py with actor\_id and model as arguments.

**python differentiate\_genre.py genre1 genre2 model**

Running the file creates output\_differentiate file which contains the tag vector for the given actor and model.

Sample output:

Actor id is: 1917810

Model is: tfidf

['brilliant', 1.0958229345381654]

['talking animals', 0.6183004819433]

['prostitution', 0.6021815039402841]

['child abuse', 0.562670125618651]

['disney animated feature', 0.5550901725201548]

['teenagers', 0.5506283315741418]

['dramatic', 0.538537323502579]

['funny', 0.5363518785340825]

['disney', 0.5211651612848374]

['heist', 0.4987266130364422]

['fun movie', 0.49741796670487726]

**Conclusion**

Created to programs to print tag vectors with weights for actors, genres and users using tf or tf-idf model. Created a program to calculate differentiating tag vector between two genres.

**Bibliography**

1. G. Salton and C. Buckley. On the use of spreading activation methods in automatic information retrieval. In SIGIR ’88: Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 147–160, 1988a.
2. <http://www.tfidf.com/>
3. K. Sparck Jones. "A statistical interpretation of term specificity and its application in retrieval". Journal of Documentation, 28 (1). 1972.
4. G. Salton and Edward Fox and Wu Harry Wu. "Extended Boolean information retrieval". Communications of the ACM, 26 (11). 1983.
5. G. Salton and M. J. McGill. "Introduction to modern information retrieval". 1983
6. G. Salton and C. Buckley. "Term-weighting approaches in automatic text retrieval". Information Processing & Management, 24 (5). 1988.
7. H. Wu and R. Luk and K. Wong and K. Kwok. "Interpreting TF-IDF term weights as making relevance decisions". ACM Transactions on Information Systems, 26 (3). 2008.