Web Information Retrieval - Homework 2

Simone Conia, Marco Favorito, Shani Dana Guetta

May 25 2017

Contents

1 Homework Overview

The objective of this homework is to focus on link-analysis techniques and in particular on the Topic-specific PageRank algorithm.

1.1 MovieLens 100k Dataset

In this homework we will work with data obtained from the MovieLens 100k Dataset, a set of 100,000 ratings from 1000 users on 1700 movies.

For more details about MovieLens visit:
https://grouplens.org/datasets/movielens/100k/

1.1.1 Movie-Graph

All the parts of this homework will work on the file ”movie_graph.txt” which represents a weighted and undirected graph. This file contains 1682 movies_ids which are the graph nodes, and 983206 relations between movies which are the graph edges.

The file “movie_graph.txt” represents one edge per line according to the following format:

```
Movie_ID    Movie_ID    Weight
  ...         ...       ...
```

The lines are sorted by ascending order of the first column and then by the second column.

1.1.2 User-Movie-Rating-Data

The file “user_movie_rating.txt” used in the second part of the homework contains real data about user ratings on movies.

The file contains 100000 ratings of 942 users, one rating per line, in the following format:

```
User_ID    Movie_ID    Rate
  ...        ...       ...
```

The rows of the file are grouped by userID.

1.1.3 Movie-Categories-Data

The file “category_movies.txt” used in the third part contains data about movies categories. Categories_IDs corresponds to line numbers of the file, and each line is a list of movies_IDs which belong to that category. In total we have five categories, with id in [1, 5].

2 Topic-Specific PageRank

The first part of the homework required to implement a Topic-specific PageRank algorithm.

2.1 Introduction

Assuming a random surfer model and given $0 < \alpha < 1$, at each iteration of the standard PageRank algorithm, the random surfer jumps with probability $1 - \alpha$ to a node chosen uniformly at random. When this happens, we say that the random surfer teleported to this node.

In the topic-specific PageRank algorithm, the random surfer teleports only to a subset of the nodes of the original graph, that is, the node the surfer teleports to is not chosen uniformly at random. This subset is indeed the topic the user is interested in. The topic-specific PageRank algorithm returns a PageRank vector that is biased towards the topic the user is interested in.

It can be proved that, provided a non-empty set S of a topic-related nodes, it follows that there is a non-empty set of nodes $Y \supseteq S$ over which the random walk has a steady-state distribution.

2.2 Implementation

Our implementation of the Topic-specific PageRank algorithm is based on the standard random-surfer-based PageRank algorithm. The topic of interest is represented as a dictionary of $(node, bias)$. For each node, the probability the random surfer teleports to that node is proportional to the *bias* of that node. If the *bias* is 0, it means the corresponding node doesn't belong to the topic of interest.

From a high-level point of view, this algorithm is implemented in the following way:

Listing : TopicSpecificPageRankIteration.txt

```
topic_specific_pagerank_iteration(graph, page_rank_vector,
    teleporting_distribution, ...)

    ...

    # Compute the leakedPR subtracting all the computed scores
  leakedPR = 1.
  for node in graph:
    leakedPR -= r[node]

  # Distribute the teleporting probability
```

```
    for node in graph:
      next_page_rank_vector[node] = r[node] + leakedPR *
       teleporting_distribution[node]

    ...

    return next_page_rank_vector
```

Note: the teleporting_distribution dictionary is indeed a distribution, so the sum of its values should be equal to 1.

3 Movie recommendation based on user ratings

The goal of this part is to implement a method for recommending movies to users based on their ratings as expressed in the file "user_movie_rating.txt".

3.1 Introduction

The recommendation must be performed by applying Topic-specific PageRank on Movie-Graph, by considering as nodes of the topic all movies rated by the input user. To increase the personalization of the recommendation, the teleporting probability distribution among all nodes in the virtual topic must be biased using the rating values that the input user has given to each rated movies.

3.2 Implementation

For this part of the homework, we can use the Topic-specific PageRank algorithm we developed in the previous section. In particular, in this case we define the topic as the set of movies rated by the user of interest. Therefore we build a dictionary of (*node*, *teleporting_bias*), where the *teleporting_bias* = *rating(node) / sum(ratings)*.

The resulting PageRank vector must be filtered by removing all the movies the user has already seen, and sorted by descending value of PageRank.

4 Movie recommendation based on user preferences

This part of the homework requires to implement a method that, given a set of categories and the preferences a user has over these categories, recommends some movies.

4.1 Introduction

The main point of this part is that we are not allowed to compute the PageRank vector at recommendation-time. For this reason, we have to split the computation in an offline part and an online part. From a high-level point of view, we have to find a way to compute the PageRank vector offline, and then use the result to compute recommendations online whenever a user preference vector comes in.

4.2 Implementation

We can take advantage of the fact that we can express a user PageRank vector as a linear combination of the PageRank vectors of the topics the user is interested in. Suppose we have a user u_i whose normalized preferences vector for n movie categories is the following one:

$$u_i = (u_{i_1}, u_{i_2}, \cdots, u_{i_n})$$

For each category c, we can compute the corresponding Topic-specific PageRank vector π_c:

$$\pi = (\pi_1, \pi_2, \cdots, \pi_n)$$

Then the PageRank vector that takes into account the preferences of u_i can be computed as follows:

$$\pi_{u_i} = \sum_{k=1}^{n} \pi_k \cdot u_{i_k}$$

It follows that we need the offline part to compute a PageRank vector for each category and store these vectors in some files. The online part will make use of these pre-computed PageRank vectors to easily compute the recommended movies. Notice that the online part also has to filter the movies the user has already seen, and sort the recommendations by descending value of their score.

5 Movie recommendations for a group of users

In this last part of the homework, it is requested to develop a method to recommend movies to a group of users.

5.1 Introduction

The main point of this part is that the group is composed by heterogeneous users who have different relative importance. Intuitively, we can develop a movie recommendation system that takes into consideration all the movies rated by all the users of the group.

One of the main points of this last part is that every user of a group may have a different relative importance with respect to the other users. If we manage to represent all the user ratings with a single data structure that also takes into account user importance levels, then we can compute the Topic-specific PageRank just once to obtain our recommendations.

5.2 Implementation

The implementation of this movie recommendation system can be divided into two parts. In the first part, we build a single data structure that represents the group while also considering the different importance level among the users.

Listing : MergeDistributions.txt

```
merge_distributions(dist_list, weights):

  result = {}

  if sum(weights)!=1.:
    weights = [w/sum(weights) for w in weights]

  for distribution, weight in zip(dist_list, weights):
    for id in distribution:
      if id in result:
        result[id] += distribution[id]*weight
      else:
        result[id] = distribution[id]*weight

  return result
```

In the second part, we use the merged distribution (together with the movie graph) as the input of the Topic-specific PageRank.

As we have seen in the previous sections, the result must be filtered to remove movies already seen by any of the user in the group, and sorted by descending value of recommendation score.