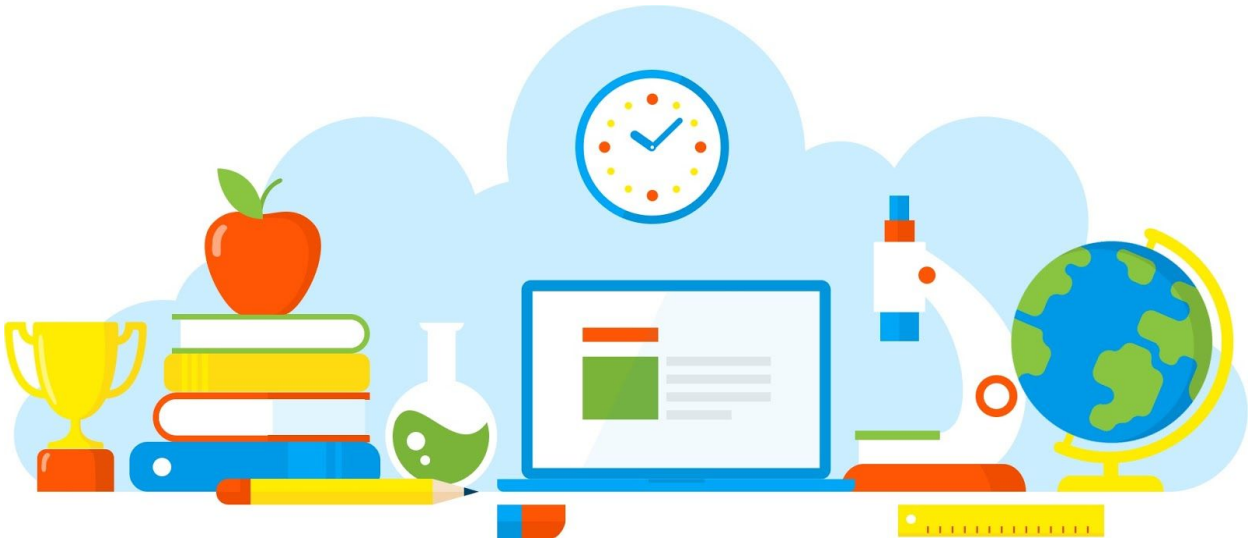

GRank

Teacher: Dr Haratizadeh

Mahdi Sadeghi
1398/9/12

بارگذاری و پیش پردازش داده‌ها

پس بارگذاری دیتاست در حافظه توسط تابع read_csv، یک ورودی T از کاربر گرفته میشود تا با استفاده از آن، از هر کاربر T تا rating انتخاب و در دیتاست آموزش قرار داده میشود. همچنین کاربرانی که کمتر از T+10 نظر ثبت شده دارند به طور کلی از دیتاست حذف میشوند. همچنین دو لیست جدا شامل کل یوزرها و کل فیلم ها هم از دیتاست استخراج میشود تا در مراحل بعد استفاده کنیم.

پس از آنکه دیتاست لود شد، آن را تبدیل به یک ماتریس اسپارس میکنیم. این کار در مراحل بعد به ما کمک میکند که به راحتی بتوانیم امتیازی که یک کاربر به یک فیلم داده است را بدست آوریم. این ماتریس به صورت یک دیکشنری ذخیره شده است که هر یوزر در آن یک آرایه شامل تیوپل‌های ریتینگش دارد.

```
def generate_ratings_matrix(ratings_dataset):
    """
    in this function we convert the ratings_dataset to a matrix with
rows identifying users
    and the columns as movies. value of element i,j in the matrix
address the rating given
    by user i to the movie j. the matrix is saved in memory as sparse
matrix.
    :param ratings_dataset:
    :return: sparse rating matrix
    """
    ratings_matrix = {}
    for _, r in ratings_dataset.iterrows():
        user_id = int(r['userId'])
        movie_id = int(r['movieId'])
        rating = int(r['rating'])

        if user_id in ratings_matrix:
            ratings_matrix[user_id].append((movie_id, rating))
        else:
            ratings_matrix[user_id] = [(movie_id, rating)]

    return ratings_matrix
```

ساخت گراف TPG

در ابتدا به ازای هر یوزر یک نود ساخته میشود. پس از آن برای هر فیلم دو گره، یکی برای desirable و یکی برای undesirable میسازیم. در نهایت تمام دوتایی‌های ممکن از فیلم‌ها که به صورت A > B است را به صورت گره در گراف میسازیم و هر کدام را به گره‌های desirable و undesirable مختص خود متصل میکنیم.

پس از تکمیل ساخت گره‌ها نوبت اتصال کاربران به گره‌های دوتایی است. در این مرحله به ازای هر نظر کاربر، آن را با سایر نظرات کاربر مقایسه میکنیم و در صورتی که نظر اول (A) امتیاز بالاتری نسبت به نظر دوم (B) داشت، آن را به A>B متصل میکنیم. در غیر اینصورت به B>A متصل میشود. در حالتی که امتیاز دو فیلم یکی بود، یوزر به هیچکدام از این گره‌ها متصل نمیشود.

در این برنامه جهت ساخت و کار با گراف‌ها از کتابخانه‌ی networkx استفاده شده است.

```
def generate_tpg(ratings_matrix, user_ids, movie_ids):
    tpg = nx.Graph()

    # for each user, create a node
    for user_id in user_ids:
        print('\t> creating user node {0}'.format(user_id))
        user_node = 'user_{0}'.format(user_id)
        if not tpg.has_node(user_node):
            tpg.add_node(user_node)

    # create desirable and undesirable nodes foreach movie
    for movie_id in movie_ids:
        print('\t> creating movie nodes for movie
{0}'.format(movie_id))
        item_d_node = 'item_{0}_d'.format(movie_id)
        item_u_node = 'item_{0}_u'.format(movie_id)

        if not tpg.has_node(item_d_node):
            tpg.add_node(item_d_node)

        if not tpg.has_node(item_u_node):
            tpg.add_node(item_u_node)

    # create A > B pairs for each possible tuple of movies
    for i in range(0, len(movie_ids)):
        movie_id_1 = movie_ids[i]
        item_d_i_node = 'item_{0}_d'.format(movie_id_1)
```

```
        item_u_i_node = 'item_{0}_u'.format(movie_id_1)

        for j in range(i + 1, len(movie_ids)):
            movie_id_2 = movie_ids[j]
            item_d_j_node = 'item_{0}_d'.format(movie_id_2)
            item_u_j_node = 'item_{0}_u'.format(movie_id_2)

            pair_1_node = 'pair_{0}>{1}'.format(movie_id_1,
movie_id_2)
            pair_2_node = 'pair_{0}>{1}'.format(movie_id_2,
movie_id_1)

            if not tpg.has_node(pair_1_node):
                tpg.add_node(pair_1_node)
                tpg.add_edges_from([(pair_1_node, item_d_i_node),
(pair_1_node, item_u_j_node)])

            if not tpg.has_node(pair_2_node):
                tpg.add_node(pair_2_node)
                tpg.add_edges_from([(pair_2_node, item_d_j_node),
(pair_2_node, item_u_i_node)])

    for user_id in ratings_matrix:
        user_ratings = ratings_matrix[user_id]

        if user_id == 0:
            continue

        print('\t> generating graph of user {0}'.format(user_id))
        user_node = 'user_{0}'.format(user_id)

        # get all the movies that specific user has rated and create a
node for each pair of them based on a constraint
        # that which movie he rated has a higher rate than the other.
we also connect the pair nodes to their respective
        # nodes in the items section of the graph (for example "A > B"
will be connected to "A_Desirable" and
        # "B_UnDesirable" respectively
```

```
        for i in range(len(user_ratings)):
            movie_id_1 = user_ratings[i][0]
            movie_rating_1 = user_ratings[i][1]
            for j in range(i + 1, len(user_ratings)):
                movie_id_2 = user_ratings[j][0]
                movie_rating_2 = user_ratings[j][1]

                if movie_rating_1 > movie_rating_2:
                    pair_node = 'pair_{0}>{1}'.format(movie_id_1,
movie_id_2)
                elif movie_rating_2 > movie_rating_1:
                    pair_node = 'pair_{0}>{1}'.format(movie_id_2,
movie_id_1)
                else:
                    continue

                # finally connect the user node to the pair node
                tpg.add_edge(user_node, pair_node)

    return tpg
```

یافتن فیلم‌های مرتبط با کاربر

برای یافتن top k recommendation برای یک کاربر، ابتدا یک pagerank با استفاده از وکتور شخصی سازی که تنها درایه‌ی آن یوزر در آن یک است میسازیم. پس از آن میزان GRank را برای تمام فیلم‌ها محاسبه کرده و k تا از فیلم‌ها با بالاترین امتیاز را بر میگردانیم.

```
def compute_pagerank(tpg, user_id):
   personalization_matrix = {'user_{0}'.format(user_id): 1}
   # the matrix is generated using networkx library
   matrix = nx.pagerank(tpg, alpha=0.85,
personalization=personalization_matrix)

   return matrix

def get_top_k_recommendations(pagerank, movie_ids, k):
   movie_granks = []
   for movie_id in movie_ids:
       grank = _compute_grank(pagerank, movie_id)
       movie_granks.append((grank, movie_id))

   movie_granks = sorted(movie_granks, key=lambda x: x[0],
reverse=True)
   return movie_granks[:k]
```