RECOMMENDATIONS BASED ON SEQUENCES

Hoang Duc Manh, MAT.180387
ducmanh.hoang@studenti.unitn.it

Scenario

Nowadays, online movie becomes more and more ubiquitous. We can see the movies online right our computers, mobile devices. It becomes a convenient way to watch our favorite movies or famous movies at some time. For company side, they try to provide their online users the best service. They try to understand the users' preferences to suggest for their users the suitable, preferable movies.
As many users' transaction datasets, people do not always watch movie randomly, they watch movies by some order or preferences. For instance, there are some movies on store (m_1, m_2, m_3, v_1, v_2, v_3, v_4, f_1, f_2, f_3) and 3 users (u_1, u_2, u_3) on the system. User u_1 saw <m_1, m_2, f_1, f_2, f_3, m_3>. User u_2 saw <m_1, m_2, m_3, v_1, v_2>. User u_3 saw <m_1, m_2, f_1, f_2>. As we see, the reasons leading to the various behavior sequences may differs between different people. However, the order of the movies watched will be effected by user's personal interests. Therefore, it is possible to find users similar interests in terms of their behavior sequences. We can see that u_1 watched m_1, m_2, m_3 and u_2 watched m_1, m_2, m_3. Intuitively, u_3 watched m_1, m_2 might like m_3 as well.

Let $M=\{m_1, m_2, ..., m_n\}$ be a finite set of movies. Let $U=\{u_1, u_2, ..., u_m\}$ be a finite set of users.
Let $m \in M$ is a movie and $u \in U$, if user u accessed movie m then $r(m)$ is the rating of user u on movie m at timestamp $t(m)$.

Defintion 1: Each user u_i $(1 \le i \le m)$ in U accessed the movies as a sequence denoted by $Su_i = \langle x_1, x_2, ..., x_l \rangle$ where $Su_i \subseteq M$, x_k $(1 \le k \le l)$ is an movie, $i \ne j$ $x_i \ne x_j$ and $(1 \le i, j \le l)$, and $i<j$ then $t(x_i)<t(x_j)$.

Definition 2: Each user u_i $(1 \le i \le n)$ in U rate on the movies sequence Su_i as a following vector $Ru_i = \langle r(x_1), r(x_2), ..., r(x_l) \rangle$ where $r(x_k)$ is the rating on x_k rated by u_i, $(0 \le r(x_k) \le 5)$ and $(1 \le k \le l)$.

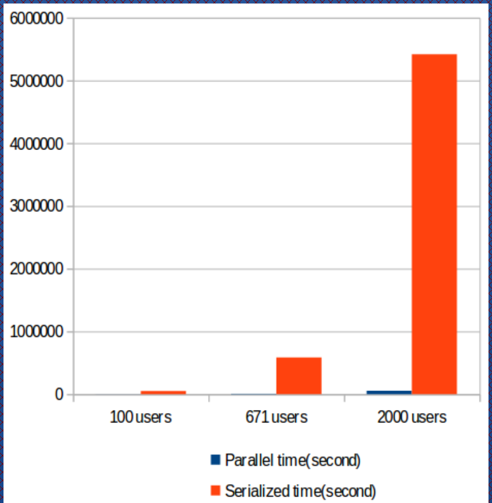
Definition 3: Each movie x_k is rated by user u_i at timestamp $t(x_k)$. Then we have the timestamp vector of rating of u_i is $Tu_i = \langle t(x_1), t(x_2), ..., t(x_l) \rangle$, x_i and x_j in the same sequence can not have the same timestamp, i.e., $t(x_i) \ne t(x_j)$ and $(i \ne j)$, and $i<j$ then $t(x_i)<t(x_j)$.

Note: the number of instances of items in a sequence is called length of the sequence.

Performance

	Parallel RMSE	Serial RMSE	Non-sequences RMSE
100 user	0.9159	0.9283	1.2668
671 user	0.9265		0.9336
2000 user	0.8939		0.8995

	Parallel time (second)	Serial time (second)
100 user	496	49349
671 user	5883	585221(estimated)
2000 user	54467	5417971(estimated)



Solution

Algorithm 2: Serialized Algorithm.
Input: training and test dataset are in format $\langle userId, movieId, rating, timestamp \rangle$ and normalized Levenshtein distance threshold σ.
Output: an evaluation metric (RMSE).

1. Initialize the sequences database $SDB = \emptyset$
2. **for** each user $u_i \in U$ **do**
3. Find the past movies sequence Su_i of user u_i on the training dataset
4. Add Su_i to the sequences database SDB
5. **end for**
6. **for** each target user $q \in U$ **do**
7. Initialize the training user list $TRLq = \emptyset$ of target user q
8. **for** each user $u_i \in U$ **do**
9. Get $Su_i \in SDB$ and $Sq \in SDB$
10. Compute normalized Levenshtein distance $ND_{(Su_i, Sq)}$ (of formula (2))
11. **if** ($ND_{(Su_i, Sq)} < \sigma$) **do**
12. Add u_i to the training user list $TRLq$ of target user q
13. **end if**
14. **end for**
15. Identify $TESTq$ and $TRAINq$ dataset depending on q and $TRLq$
16. Apply the collaborative filtering with ALS implementation for $TRAINq$ (section 4.3)
17. Predict rating for $TESTq$ (section 4.3)
18. Store the predicted result to a list for calculating the RMSE. (section 4.3)
19. **end for**
20. Compute the RMSE.

Algorithm 3: parallel algorithm.
Input: RDD training dataset denoted $trainRDD$ and RDD test dataset denoted $testRDD$ are set of tuples $\langle userId, movieId, rating, timestamp \rangle$ and normalized Levenshtein distance threshold σ.
Output: an evaluation metric (RMSE).

1. Convert $trainRDD$ to simple RDD key-value pair $\langle userId, \langle userId, movieId, rating, timestamp \rangle\rangle$, denoted $keyTrainRDD = trainRDD.keyby(lambda\, x: x['userId'])$
2. Create a dataset of $\langle userId, Iterable\langle userId, movieId, rating, timestamp \rangle\rangle$ pairs, denoted $groupByKeyTrainRDD = keyTrainRDD.groupByKey()$
3. Sort the iterable value of each element of $groupByKeyTrainRDD$ dataset based on $timestamp$ values, in order to find out the users' past movies sequences database by invoking function $groupByKeyTrainRDD.map(lambda\, x: (x[0], sorted(list(x[1]), key=getKey)))$
4. Find users' past movies sequences database (each sequence under format $\langle userId, \langle movieId_1, movieId_2, ..., movieId_n \rangle\rangle$, or in other word $\langle u_i, Su_i \rangle$) of all user denoted $SDBRDD = groupByKeyTrainRDD.map(lambda(x, y): (x, zip(*y)[0]))$
5. Get RDD Cartesian product of $SDBRDD$ and itself, denoted $cartesianSDBRDD$ containing set of RDD tuple elements (each element under format $\langle\langle userId, \langle movieId_1, movieId_2, ..., movieId_n \rangle\rangle, \langle userId, \langle movieId_1, movieId_2, ..., movieId_n \rangle\rangle\rangle$, or in other word $\langle\langle q, Sq \rangle \langle u_i, Su_i \rangle\rangle$) by calling function $SDBRDD.cartesian(SDBRDD)$
6. Compute the normalized Levenshtein distance for each element in $cartesianSDBRDD$ dataset, denoted $NDRDD = cartesianSDBRDD.map(lambda(x, y): (x[0], y[0], (levenshtein(x[1], y[1]) / float(max(len(x[1]), len(y[1]))))))$. An element of $NDRDD$ is formated as $\langle q, u_i, ND_{(Su_i, Sq)} \rangle$
7. Filter to collect the elements of $NDRDD$ having $ND_{(Su_i, Sq)} < \sigma$, denoted $filteringNDRDD = NDRDD.filter(lambda\, x: ((x[2]>0)))$
8. Map collection $filteringNDRDD$ to a RDD dataset (each tuple element under format $\langle q, u_i \rangle$), denoted $mappingNDRDD = filteringNDRDD.map(lambda\, x: (x[0], x[1]))$
9. Create tuples dataset of $\langle q, \langle u_1, u_2, ..., u_h \rangle\rangle$ pairs (where q is target user, $\langle u_1, u_2, ..., u_h \rangle$ is the training user list), denoted $CttRDD = mappingNDRDD.groupByKey().map(lambda\, x: (x[0], list(x[1])))$
10. **for** each element $\langle q, \langle u_1, u_2, ..., u_h \rangle\rangle$ **in** $CttRDD$ **do**
11. Identify $TESTq = testRDD.filter(lambda\, x: x[0] == q)$ and $TRAINq = trainRDD.filter(lambda\, x: x['userId'] \text{in} \langle u_1, u_2, ..., u_h \rangle)$ dataset
12. Apply the collaborative filtering with ALS implementation (using parallel version on Apache Spark) for $TRAINq$ (section 4.3)
13. Predict rating (using parallel version on Apache Spark) for $TESTq$ (section 4.3)
14. Store the predicted result to a list for calculating the RMSE. (section 4.3)
15. **end for**
16. Compute the RMSE.