RECOMMENDATIONS BASED ON SEQUENCES

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

ABSTRACT
This paper describes the taken procedure in order to develop an advanced recommendation system. There are many kind of algorithms for making predictions for target users, and among them Collaborative filtering are one widely adopted. However, in some domains, the user's behaviors sequences that reflect his or her preferences over items, so that users have similar behaviors sequences may likely have same similar preferences. Based on this fact, this paper introduce new approach of recommendation system by using the traditional Collaborative filtering and simultaneously combine with the Levenshtein distance in order to compute the distance measure of the item sequences. This approach will operate in order to improve the Collaborative filtering algorithm by using user behaviors item sequence distances. Moreover, this paper also present the implementation of parallel computation technology which is Apache Spark. Finally, this show you the result of the recommendation system and the advantages of performance of the using Apache Spark in data processing works.

1. INTRODUCTION

The Internet is penetrated deeply into the works, life, even mind of human. The reason I am saying that because as we see, people are trying to understand of the human behavior to find out the algorithm that can predict what is the next thing human are going to do. These system which are called recommendation system. In order to improve more and more effectively of the precision of the recommendation system, we try to improve or combine the algorithms to build a new recommendation model.

The recommendation system based on sequences is a way that is increasingly sophisticated use of data mining techniques by combining the traditional recommendation system with a distance measurement algorithm between item sequences that user present their preferences. We are going to compute the distance measure of the user's item sequences and take out number of users who have similar preferences with the target user, then predict the next interested items for the target user. We are familiar with the traditional Collaborative filtering algorithm that change entirely the entertainment industry such as movie, music, retail, etc. This method try to suggest people some thing that they might like. It is effective way to increase the number of using, buying or reducing the mistake of using.

In order to improve more and more faster and effectively of the recommendation, many ways are implemented, like inventing new compact and effective recommendation algorithm or improve the technologies to make the algorithm run faster like using distributed system, distributed computation. The system have been succeeded in applying recommendation system such as movie web page of Amazon, Netflix or Youtube which have satisfied its users, stimulated its user rely on it, bought more and more products and became loyal customers.

There are many kind of algorithm have been using to develop the recommendation system. Collaborative filtering is one of the most effective and ubiquitous technique. It uses the user based database meaning a database of the user's preferences over items that they buy, view, rate or like to produce the predictions for the next actions of the its users in order to suggest the topics, movies, products for them. Usually, Collaborative filtering implement with Alternating Least Square (ALS), Linear Regression or some similarity methods (Pearson correlation similarity or cosine similarity) that calculate the similarity between the two users or items but it did not consider sequences of the user preferences.

In some domain, the sequences of a user's behavior can reflect his or her preferences over items. Talking the following case as an example, 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

saws $\langle m_1, m_2, f_1, f_2, f_3, m_3 \rangle$. User u_2 saws $\langle m_1, m_2, m_3, v_1, v_2 \rangle$. User u_3 saws $\langle m1, m2, f1, f2 \rangle$. 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 in the u_1 's watched movies sequence contains subsequence $\langle m_1, m_2, m_3 \rangle$ and u_2 's watched movies sequence contains subsequence $\langle m_1, m_2, m_3 \rangle$. Intuitively, u_3 watched $\langle m_1, m_2 \rangle$ might like m_3 as well. Basically, there is reasonable idea to apply in this approaches in order to build a Recommender System based on Sequences.

Thus, this paper is going to introduce a new approach method to apply the sequences of preferences of users to improve the traditional Collaborative filtering method.

Because of making the Collaborative filtering more sophisticated, one other main concern is to maintain speed in processing large datasets in terms of waiting time between queries and waiting time to run the program. Spark is a technology for speeding up the computational computing software process. Spark construct on the computational cluster framework. Spark is a fast and general cluster computing system for Big Data. It provides high-level APIs in Scala, Java, Python, and R, and an optimized engine that supports general computation graphs for data analysis. It also supports a rich set of higher-level tools including Spark SQL for SQL and DataFrames, MLlib for machine learning, GraphX for graph processing, and Spark Streaming for stream processing. In general, Spark deal with the velocity, variety and volume of Big Data. In this paper we will consider Spark as a main exploiting velocity and volume technology and exploiting library of machine learning (MLlib) which is Collaborative filtering ALS.

2. RELATED WORKS

Association rule is a widely used data mining technique that generates recommendations in recommender systems [4, 6]. More specifically, the method tries to discover the relationships between product items based on patterns of co-occurrence across customer transactions. These association rule-based methods are comprised of three steps: (1) Find an association between two sets of products in a transaction database; (2) The active customer's purchase behavior is compared with the discovered association rule base to find the most similar purchasing behaviors; (3) A set of products that the active customer is most likely to purchase is then generated by selecting the top-N most commonly purchased products. However, association rule mining does not take the time stamp into account. Thus, the association rule-based recommender system cannot reflect the dynamic nature of users' behavior.

Unlike association rules, sequential patterns [2] may suggest that a consumer who buys a new product in the current time period is likely to buy another product in the next time period. While association rule discovery covers only intra-transaction patterns (itemsets), sequential pattern mining also discovers inter-transaction patterns (sequences). In essence, frequent sequential pattern discovery can be thought of as association rule discovery over a temporal database. As association rules can be used for recommendation, sequential pattern mining-based recommendation algorithms have been studied for online product recommendation in recent years [10].

The sequential pattern mining-based recommendation is to induce a predictive model from a set of examples (i.e., the sequences in database). The usefulness of sequential pattern mining-based recommendation has, to a certain extent, been demonstrated empirically by past studies on various domains such as Web browsing [10], e-commerce [3], and music [5]. However, a significant shortcoming is that these methods do not perform user-specific sequential pattern mining and, as a result, they cannot give accurate personalized recommendation to users.

Recently, Yap et al. [9] proposed a personalized sequential pattern mining-based recommendation framework. A competence score measure, considering relevance to the target user, and the sequences' recommendation ability, is used for accurate personalized recommendation. Furthermore, additional sequence knowledge is exploited to improve the efficiency and the quality of learning in sequential pattern mining algorithms. However, their framework still depends on support threshold. Thus, if some sequences do not have enough supporting samples, they cannot be used for recommendation.

3. PROBLEM STATEMENT

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 $(1\leq i\leq m)$ 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 $(1\leq k\leq l)$ is an movie, $i\neq j$ $x_i\neq x_j$ and $(1\leq i,j\leq l)$, and $i<j$ then $t(x_i)<t(x_j)$.

Definition 2: Each user u_i $(1\leq i\leq n)$ 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 , $(0\leq r(x_k)\leq 5)$ and $(1\leq k\leq l)$.

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., $t(x_i)\neq t(x_j)$ and $(i\neq j)$, 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.
For example, some instances of the dataset look like:

userId	movieId	rating	timestamp
1	31	2.5	1260759144
1	1029	3	1260759179
1	1061	3	1260759182
1	1129	2	1260759185
1	1172	4	1260759205
1	1263	2	1260759151
1	1287	2	1260759187
1	1293	2	1260759148
1	1339	3.5	1260759125
1	1343	2	1260759131
2	10	4	835355493
2	17	5	835355681
2	39	5	835355604
2	47	4	835355552
2	50	4	835355586
2	52	3	835356031
2	62	3	835355749
2	110	4	835355532
2	144	3	835356016
2	150	5	835355395

Table 1. Example of the dataset.

The above dataset table is already having timestamp following the ascending order. The timestamp column is in Unix system timestamp unit. Therefore, we have two movies sequences accessed by user having userId = 1 and userId = 2 as the follows:

$\langle m_1, v_1, v_2, v_3, f_1, f_2 \rangle$	$\rightarrow$	$\langle m_2, v_1, v_2, v_3, f_1, f_2 \rangle$	(substitution of " m_2 " for " m_1 "),
$\langle m_2, v_1, v_2, v_3, f_1, f_2 \rangle$	$\rightarrow$	$\langle m_2, v_1, v_2, v_3, m_3, f_2 \rangle$	(substitution of " m_3 " for " f_1 "),
$\langle m_2, v_1, v_2, v_3, m_3, f_2 \rangle$	$\rightarrow$	$\langle m_2, v_1, v_2, v_3, m_3, f_2, f_3 \rangle$	(insertion of " f_3 " at the end).

userId	**Accessed movies sequence**	**Rating vector**
1	<31, 1029, 1061, 1129, 1172, 1263, 1287, 1293, 1339, 1343>	<2.5, 3, 3, 2, 4, 2, 2, 2, 3.5, 2>
2	<10, 17, 39, 47, 50, 52, 62, 110, 114, 150>	<4, 5, 5, 4, 4, 3, 3, 4, 3, 5>

Table 2. Example of the movies sequences.

A sequence database SDB is a collection of sequences, i.e., $SDB=\{Su_1, Su_2, ..., Su_m\}$ where $|SDB|=m=|U|$ denotes the number of sequences (is also equal to number of users as each user has a corresponding sequence in SDB). $Su_i \in SDB$, $(i=1,2,...,m)$ is a ordered list of sequences associated with a user $u_i \in U$.

Given a target user q and past accessed movies sequence Sq and the database of all other users' past accessed movies sequences, the Recommendation System Based on Sequences task is to predict the ratings of user q on other movies that do not exist in Sq but exist in the M set. Then, it select the top number movies predicted as highest ratings in the predicted movies set. These movies correspond to the most likely interested movies list that user q will access in the future.

4. SOLUTION

4.1. Levenshtein distance

In information theory and computer science, the Levenshtein distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (i.e. insertions, deletions or substitutions) required to change one word into the other.

Definition 1: Mathematically, the Lenvenshtein distance between to strings a,b (of length $|a|$ and $|b|$ respectively) is given by $lev_{a,b}(|a|,|b|)$ where:

$$lev_{a,b}(i,j)=\begin{cases} max(i,j) if\ min(i,j)=0 \\ min\begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & otherwise \end{cases} \quad (1)$$

where $1_{(a_i \neq b_j)}$ is the indicator function equal to 0 when $a_i=b_j$ and equal to 1 otherwise, and $lev_{a,b}(i,j)$ is the distance between the first i characters of a and the first j characters of b .

Note that the first element in the minimum corresponds to deletion (from a to b), the second to insertion and the third to match or mismatch, depending on whether the respective symbols are the same.

Our problem, given a database of sequence SDB that is a collection of sequences, i.e., $SDB=\{Su_1, Su_2, ..., Su_m\}$ where Su_i $(i=1,2,...,m)$ is the u_i 's past accessed movies sequence vector and given a target user q and his/her past accessed movies sequence vector Sq .

We will covert our input in to the Levenshtein's input. In order to compute the Levenshtein distance between the two movies sequence vectors Su_i and Sq . Su_i and Sq are considered as the two strings a,b respectively. Each movie item in a sequence is considered as a character in a string.

The formulas (1) is illustrated the Levenshtein distance algorithm. We have also known how to convert the input of our problem in to Levenshtein's input. Now, we are going to apply to our problem to demonstrate working of

algorithm in detail by using an example.
For example, the Levenshtein distance between the two sequences $\langle m_{1,} v_1, v_2, v_3, f_1, f_2 \rangle$ and $\langle m_2, v_1, v_2, v_3, m_{3,} f_2, f_3 \rangle$ is 3, since the following three edits change one into the other, and there is no way to do it with fewer than three edits:

		m₁	**v₁**	**v₂**	**v₃**	**f₁**	**f₂**
	0	1	2	3	4	5	6
m₂	1	**1**	2	3	4	5	6
v₁	2	2	**1**	2	3	4	5
v₂	3	3	2	**1**	2	3	4
v₃	4	4	3	2	**1**	2	3
m₃	5	5	4	3	2	**2**	3
f₂	6	6	5	4	3	3	**2**
f₃	7	7	6	5	4	4	**3**

Table 3. Example of computing Levenshtein distance.

4.2 Normalization of Levenshtein distance.

The Levenshtein distance between two watched movies sequences is a value that show the minimum number of editing operation (i.e. insertions, deletions or substitutions) required to change one sequence in to the other.
Our problem, given a database of sequence SDB that is a collection of sequences, i.e., $SDB=\{Su_1, Su_2, \ldots, Su_m\}$ where Su_i $(i=1,2,\ldots,m)$ is the u_i 's past accessed movies sequence vector and given a target user q and his/her past accessed movies sequence vector Sq . The length of the two sequence Su_i and Sq is $|Su_i|$ and $|Sq|$
The Levenshtein distance will be $0 \leq lv \leq max(|Su_i|, |Sq|)$ Thus, After we found the Levenshtein distance between two sequences. We need to normalize the Levenshtein distance. The normalized Levenshtein distance is denoted $ND_{(Su_i, Sq)}$. In order to normalize Levenshtein distance we divide it for the maximum value between the two lengths of two sequences.

$$ND_{(Su_i, Sq)} = \frac{lev_{Su_i, Sq}(i,j)}{max(|Su_i|, |Sq|)} \qquad (2)$$

where $ND_{(Su_i, Sq)}$ is normalized distance value of the Levenshtein distance of the two sequences Su_i and Sq . $lev_{Su_i, Sq}(i,j)$ is the Levenshtein distance of the two sequences Su_i and Sq . $max(|Su_i|, |Sq|)$ is the maximum value of lengths of the two sequences Su_i and Sq .
The normalized Levenshtein distance will help us to identify how much is different between the two sequences Su_i and Sq better. In other hand, It is the weight distance between the two sequences Su_i and Sq .
The value of normalized Levenshtein distance is in the range as $0 \leq ND_{(Su_i, Sq)} \leq 1$. $ND_{(Su_i, Sq)}=0$ means the two sequences Su_i and Sq to be identical. $ND_{(Su_i, Sq)}=1$ means the two sequences Su_i and Sq to be completely different. In our approach, we will consider the sequences have $ND_{(Su_i, Sq)} \neq 1$.

4.3. Collaborative Filtering with Alternating Least Square implementation

Alternating Least Square (ALS) with weighted-λ -regularization described in section 3.1 of the paper Large-scale Parallel Collaborative Filtering for the Netflix Prize [11]. They formulated the matrix completion problem as a matrix-completion-by-factorization problem. They minimize the empirical, total loss which is simply the everage of the square loss function defined as the squared error:
$l^2(r,u,m)=(r-(u,m))^2$ where r number of latent factor, u number of users in the dataset, m number of movies in dataset.
They regularize the empirical loss with a regularization term similar to Tikhonov regularization. The function is:

$$f(U,M)=\sum_{(i,j)\in I}(r_{ij}-u_i^T m_j)^2+\lambda(\sum_i n_{u_i}\|u_i\|^2+\sum_j n_{m_j}\|m_j\|^2)$$

They then derive $u_i=A_i^{-1}V_i$ and $m_i=A_j^{-1}V_j$ where $A_i=M_{I_i}M_{I_i}^T+\lambda n_{u_i}E, V_j=M_{I_i}R^T(i,I_i)$ and $A_j=U_{I_j}U_{I_j}^T+\lambda n_{m_j}E, V_j=U_{I_j}R^T(j,I_j)$ where E is the $r\,x\,r$ identify matrix.

The pseudo code for the algorithm described in the paper and class notes can be summarized:

1. Initialize pattern matrix M by assigning the average rating for movies to the first row of M and small random numbers for the remaining entries. Initialize elements of U to small random numbers
2. Fix M, solve U by minimizing the objective function (sum of squared errors).
3. Fix U, solve M by minimizing the objective function.
4. Repeat Steps 2 and 3 until the algorithm has converged or a stopping criterion is satisfied.

4.4. Algorithm Description

This part is going to introduce a new algorithm for Recommendation based on sequences problem. The idea is that we will use Collaborative Filtering with Alternating Least Square implementation to build a recommendation system. In order to meet the project requirement, we are deciding to apply the Sequence method at some point in the Collaborative Filtering method which purpose to combine Collaborative Filtering and Sequence method together. Based on the above discussion, the following outline step is the general pseudo code illustrated the step of the works we are going to do.

Algorithm 1: General preliminary of the algorithm

Input: the raw dataset contain $\langle userId, movieId, rating, timestamp\rangle$. Actually, this input provide M is finite set of movies, U is finite set of users, Su_i is the accessed $movies\subseteq M$ sequence of user $u_i\in U$, Ru_i is the rating vector of user $u_i\in U$ on $Su_i\subseteq M$, Tu_i is the timestamp vector of rating of user u_i on Su_i, and SDB is the collection of movies sequences. Addition, σ is defined as the threshold of normalized Levenshtein distance value.

Output: instead of outputting the ratings of all user on the movies $m_i\in M$ (M is the finite set of movies items), the algorithm will output a evaluation metrics (RMSE)

```
1. Separate dataset into two set training and test set such that $U\ of\ training\ set=U\ of\ test\ set=U$ , $M\ of\ training\ set\cup M\ of\ test\ set=M$ .
2. Find past accessed movies sequences database $SDB$ of all user $u_i\in U$ on training set, $Su_i\subseteq M$ and $Su_i\in SDB$ on training set.
3. Compute normalized Levenshtein distance $ND_{(Su_i, Sq)}$ (at formula $(2)$ ) of each target user's past movies sequence $Sq\in SDB$ with the other users' past movies sequences $Su_i\in SDB$ .
4. Create collection denoted $Ctt$ of key pair value in format $\langle q,\langle u_1,u_2,\dots,u_h\rangle\rangle$ where $q$ is key (target user) and value is $\langle u_1,u_2,\dots,u_h\rangle$ is the set of users such that $ND_{(Su_i, Sq)}$ < $\sigma$ .
5. Identify test dataset and training dataset of each target user based on each element of $Ctt$ . The test dataset $TESTq$ of target user $q$ is taken from test dataset with only dataset of $q$ . The training dataset $TRAINq$ of target user $q$ is taken from training dataset with only dataset of users contain in $\langle u_1,u_2,\dots,u_h\rangle$ set .
6. Build training model for target user $q$ by using Collaborative Filtering with ALS implementation described in section 4.3 on training set $TRAINq$ and test training model on test set $TESTq$ .
7. Evaluation metrics (RMSE )
```

This above algorithm describe step by step of the main process of our recommendation system. We have to divide the dataset into two set, to be training set and test set. On the training set we find out all movies sequences of all users. In other word, finding out the all movies sequences of all user in training set is the finding out users' past movies sequences. In the traditional recommendation system algorithm, we immediately put training set to train in order to build the model for all target user i.e., the algorithm just take one time of training and one time for testing. But, in this approach we will consider the sequences in the training dataset in order to build a personalized training set of user's interest based on sequences. To do so, we next apply the normalized Levenshtein distance (formula (2)) for each target user's past sequence with all users' past sequences in order to seek the individual training set for each target user. Then, we using the traditional collaborative filtering with ALS implementation algorithm (described in section 4.3) to build the model and test model.

Because of the Big Data concern, meaning that we will have to deal with big dataset whereas the idea of Recommendation based on sequences problem will make the Recommendation algorithm more sophisticated. We are considering sequences i.e., the algorithm of traditional recommendation implemented Collaborative Filtering with Alternating Least Square implementation, at the first step of process, we need to add the sequence considering in order to find out the each user's individual training dataset. Then, each user's individual training dataset will be trained to build individual model for respective user. These models are created by using Collaborative Filtering method. In Data Mining topic, we are going to use the serialized program version to show the feasibility of our approach. We are also going to design an algorithm for parallel program version accounted to Big Data problem. The parallel program version will work in parallel, it will improve the performance and speed up processing of Recommender System. We assume that step 1 on the Algorithm 1 is already processed to create training and test set, then we do not consider in the two following algorithm version.

4.4.1. Algorithm Description for Serialized Version

Based on the Algorithm 1 discussion, the proposed of serialized algorithm is described in Algorithm 2.

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.
```

The dataset is going to use for Algorithm 2 as in format $\langle userId, movieId, rating, timestamp\rangle$. Otherwise, we can use some other datasets which can transform to compatible format contain users, items, ratings of users for items and the most important thing to be timestamp which give us to know when a user rated for an item. It is important because It will help us to identify the sequences of the items. The output of the Algorithm 2 is an evaluation metric RMSE that can show how is real values of rating and predicted values of rating close, in order to see the precision of our recommendation system. Step 1 to 5, Algorithm 2 find the past movies sequences Su_i of all users $u_i \in U$ $(i=1,2,...,m)$. To do so, we using the vectors of timestamp of accessed movies sequences Tu_i. We collect all sequences Su_i into the sequences database SDB. Step 6 to 14, in order to find out the personalized training dataset for target user q, for each target user q, we take the past movies sequence $Sq \in SDB$ of q, and then compute the normalized Levenshtein distance $ND_{(Su_i, Sq)}$ (formula (2)) between Sq and one by one $Su_i \in SDB$. Simultaneously, we check the value of $ND_{(Su_i, Sq)}$. If $ND_{(Su_i, Sq)} < \sigma$ then we store the user u_i in the training user list of target user q, $TRLq$. Step 15 to 19, we now use the pair key value to identify the training and test dataset of target user q ($TRAINq$, $TESTq$). The test dataset depends on the key and the training dataset depends on the value part containing the list of training user of target user q. Finally, we apply the collaborative filtering with ALS implementation for training and testing (in description 4.3).

4.4.1. Algorithm Description for Parallel Version

Based on the Algorithm 1 and Algorithm 2 discussion, the proposed of parallel algorithm is described in Algorithm 3. Algorithm 3 has been implemented on Apache Spark, based on RDDs. When a map operation is run on an RDD, the function is applied in parallel to all of its elements.

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).

```
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 $\langle q, \langle u_1, u_2, \ldots, u_h\rangle\rangle$ pairs (where $q$ is target user,
    $\langle u_1, u_2, \ldots, u_h\rangle$ is the training user list), denoted
    $CttRDD = mappingNDRDD.groupByKey().map(lambda\, x:(x[0], list(x[1])))$
10. for each element $\langle q, \langle u_1, u_2, \ldots, u_h\rangle\rangle$ in $CttRDD$ do
11.         Identify $TESTq = testRDD.filter(lambda\, x : x[0] == q)$ and
            $TRAINq = trainRDD.filter(lambda\, x : x['userId'] \text{in} \langle u_1, u_2, \ldots, u_h\rangle)$ 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.
```

Algorithm 3 is given inputs to be RDD training dataset denoted $trainRDD$ and RDD test dataset denoted $testRDD$, both to be in format $\langle userId, movieId, rating, timestamp\rangle$ and a normalized Levenshtein distance threshold σ . The output of the Algorithm 3 is an evaluation metric RMSE that can show how is real values of rating and predicted values of rating close, in order to see the precision of our recommendation system. Step 1, the algorithm start with transforming $trainRDD$ dataset into a format as key-value pairs $\langle userId, \langle userId, movieId, rating, timestamp\rangle\rangle$ dataset, denoted $keyTrainRDD$. Step 2, based on $keyTrainRDD$, the tuple elements will be grouped by their key to result a RDD dataset denoted $groupByKeyTrainRDD$ in format $\langle userId, Iterable\langle userId, movieId, rating, timestamp\rangle\rangle$. Each $groupByKeyTrainRDD$'s element contain $u_i \in U$ is a user and $Iterable\langle userId, movieId, rating, timestamp\rangle$ is the list u_i 's past movie, but it is not order following timestamp. Step 3, therefore, we have to sort the iterable value to have order of accessing movies of a user. Sorting on timestamp yields the accessed movies sequence of user u_i . Step 4, in order to create users' past movies sequences, each sorted $groupByKeyTrainRDD$'s element is implemented mapping into a RDD element in format $\langle userId, \langle movieId_1, movieId_2, \ldots, movieId_n\rangle\rangle$, or in other word $\langle u_i, Su_i\rangle$. At this point, we are having the movies sequences database $SDBRDD$ by collecting the result at Step 4. Step 5, we take RDD Cartesian product of $SDBRDD$ and itself to create a RDD dataset called $cartesianSDBRDD$. Each element of $cartesianSDBRDD$ is in format $\langle\langle q, Sq\rangle\langle u_i, Su_i\rangle\rangle$. Step 6, we compute the normalized Levenshtein distance $ND_{(Su_i, Sq)}$ for each element of $cartesianSDBRDD$, then we get a RDD database call $NDRDD$ that have each element in format $\langle q, u_i, ND_{(Su_i, Sq)}\rangle$ where $q \in U$ is the target user, $u_i \in U$ is training user, Su_i , Sq is the past movies sequences respectively, and $ND_{(Su_i, Sq)}$ is the normalized Levenshtein distance of Su_i , and

Sq sequences. Step 7, we use the normalized Levenshtein threshold σ to filter out and keep the pair sequences that have $ND_{(Su_i, Sq)} < \sigma$. The RDD dataset after Step 7 is called $filteringNDRDD$. Step 8, on filteringNDRDD dataset, we map each element of it into pair of $\langle q, u_i \rangle$, named $mappingNDRDD$. Step 9, Create RDD database of tuple pairs target user q and training users list of it, $\langle q, \langle u_1, u_2, \ldots, u_h \rangle \rangle$, called $CttRDD$. Finally, for each element in $CttRDD$, we use the key-value pair $\langle q, \langle u_1, u_2, \ldots, u_h \rangle \rangle$ to identify the training and test dataset of target user q ($TRAINq$, $TESTq$). The test dataset depends on the key and the training dataset depends on the value part containing the list of training user of target user q . Finally, we apply the collaborative filtering with ALS implementation (implemented in Apache Spark, parallel version) for training and testing on datasets $TRAINq$ and $TESTq$.

5. PERFORMANCE EXPERIMENTS

We are going to present two part evaluations of two proposed algorithm versions (parallel and serialized algorithm versions). Then, we compare the time effectiveness of implementing in different ways (parallel and serialized algorithm). Both of algorithms will result the Root mean square error RMSE between values predicted by ALS model and the values actually observed in order to get the measure of accuracy. We also implemented a parallel program version but in non sequences algorithm version in order to compare the RMSE measurement on the different size of dataset.

The experiments were performed on a computer with Intel® Core™ i3-3217U CPU @ 1.80GHz × 4 , 3,7 GiB of RAM and running on Ubuntu 14.04

We used the MovieLens dataset from GroupLens project (http://grouplens.org/datasets/movielens/) for the performance evaluation. It captured the rating and rating time of users on movies. Otherwise, we can use some other datasets which can transform to compatible format contain $\langle userId, movieId, rating, timestamp \rangle$ (see an example in Table 1).

We are going to use 3 different size of datasets which are 100 users, 671 users, 2000 users dataset to have the best understanding of time performance of both algorithm versions. Looking at inside the datasets, the average of items in each user's past movies sequence is 150 items meaning $|Su_i| = 150$ in average.

For the both algorithm versions, we set the normalized Levenshtein distance threshold parameter $\sigma = 1$.

For the ALS model, we set value for the parameters λ , r , and K . For regularization parameter we set $\lambda = 0.1$. Convergence criterion is set $K = 5$. Dimensionality of latent feature space $r = 12$. These parameters were determined by using the non-sequence parallel version to test the dataset and achieve the best parameters for the datasets.

For the serialized version, the testing time take so long. Thus, we will give the real result of time consuming on 100 users dataset and the other ones using estimated by time with the rate of the number time training and testing of each user.

For the configuration of the running machine, we used python 2.7 and spark-2.0.2-bin-hadoop2.7 to run our programs in experiment. The serialized version just need to run on python 2.7. The parallel version need to configure the spark-2.0.2-bin-hadoop2.7 system. Our Spark system was configured by one master and one slaver. The master and slaver is connected on only above computer system, meaning that, we have one local slaver.

This is the RMSE table of experiments:

	Parallel RMSE	Serialized RMSE	Non-sequences RMSE
100 users	0.9159	0.9283	1.2668
671 users	0.9265		0.9336

2000 users	0.8939		0.8995

Table 4. RMSEs of running experiments on different datasets and algorithm versions.

In Table 4, the evaluation metrics show that our algorithm give us the better recommendation, mean that combining of sequences with a traditional collaborative filtering approach is feasible. It increase the accuracy of the recommender system.
This is the time table of experiments:

	Parallel time (second)	Serialized time (second)
100 users	496	49349
671 users	5883	585221(estimated)
2000 users	54467	5417971(estimated)

Table 5. Running time of experiments on different datasets and algorithm versions.

In Table 5, we see that the time to run dataset of 100 users on serialized version is higher than on parallel version 99.4 times. The other parallel experiment using 2000 users dataset consume 54467 second while serialized experiment using 100 users dataset consume 49349 second. The running time table show that, our algorithm benefits greatly from parallel computation. As expected, the parallel computation speedup our algorithm and deal with scalability problem.
Because of greatly consuming time of serialized version, thus experiment cannot perform on the 671 users and 2000 users dataset. Therefore, we estimated time consuming of serialized version by multiplying 99.4 times of parallel version's time consuming. The comparison of time consuming on different version can be shown in Figure 1.

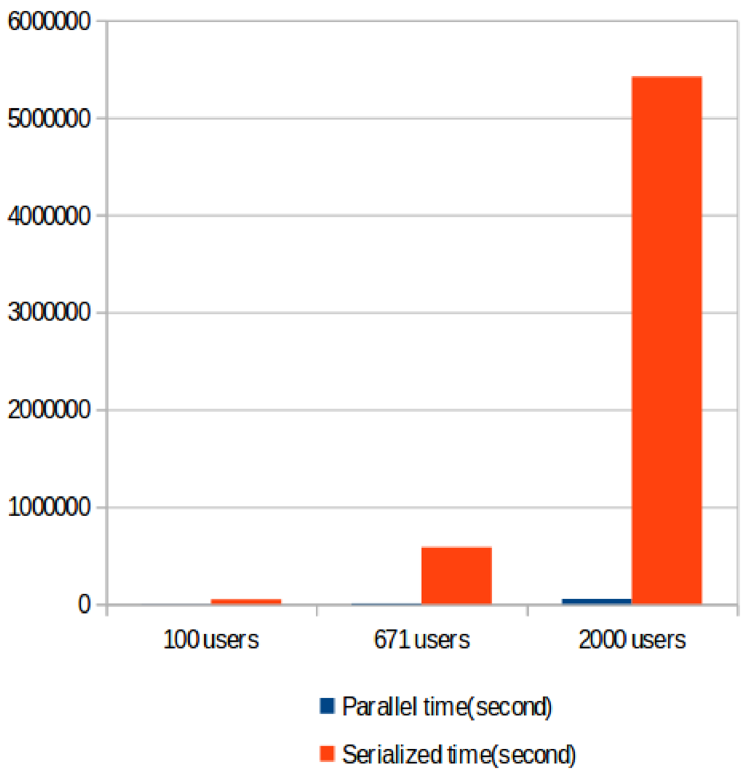

Figure 1. Evaluation of time performances between parallel and serialized version.

6. CONCLUSION

In this paper we introduced a new approach of designing a recommendation system called Recommendation based on sequences. We informally and formally defined it and described how it is applied in the algorithms. The algorithms basically consist of main steps, (1) finding the sequences of users' past accessed movies, (2) computing the Levenshtein distance of the pairs of target and training users, (3) using Collaborative filtering with ALS implementation to build movie recommender system. In terms of Data Mining and Big Data, we also design two algorithms, one of serialized version and one of parallel version. Both of them were implemented and gave us positive results. In Data Mining part, both algorithm gave us the evaluation RSMEs are better than the traditional recommender system giving us. Because of making more sophisticated the traditional recommendation system, our algorithm needed more computation resources and make our program take long time to run. Thus, we also presented the solution for that by using Big Data technique that used the distributed and parallel computing Apache Spark. The algorithm of parallel computing also described. It implementing shows an greatly benefit on time consuming by addressing the parallel version is faster 99.4 times than the serialized version. Thus, our approach on Recommendation based on sequences topic is feasible solution.

Acknowledgments: This work was supported by prof. Yannis Velegrakis, who taught me Data Mining and Big Data courses and gave me idea of this project. I would like to thank Daniele Foroni and Paolo Sottovia, who is assistants of prof. Yannis Velegrakis. They were helping me in technique questions of implementing tools.

7. REFERENCES

[1]. Agarwal D, Chen B-C, Elango P, Ramakrishnan R (2013) Content recommendation on web portals. Commun ACM 56:92–101

[2]. Febrer-Hernández JK, Palancar JH (2012) Sequential pattern mining algorithms review. Intell Data Anal 16:451–466

[3]. Huang C-L, Huang W-L (2009) Handling sequential pattern decay: developing a two-stage collaborative recommender system. Electron Commer R A 8:117–129

[4]. Kazienko P, Pilarczyk M (2008) Hyperlink recommendation based on positive and negative association rules. New Generation Comput 26:227–244

[5]. Liu N-H (2013) Comparison of content-based music recommendation using different distance estimation methods. Appl Intell 38:160–174

[6]. Paranjape-Voditel P, Deshpande U (2013) A stock market portfolio recommender system based on association rule mining. Appl Soft Comput 13:1055–1063

[7]. Park DH, Kim HK, Choi IY, Kim JK (2012) A literature review and classification of recommender systems research. Expert Syst Appl 39:10059–10072

[8]. Pera MS, Ng Y-K (2013) A group recommender for movies based on content similarity and popularity. Inf Process Manage 49:673–687

[9]. Yap G-E, Li X-L, Yu PS (2012) Effective next-items recommendation via personalized sequential pattern mining. In: Proceedings of 17th international conference on database systems for advanced applications, pp 48–64

[10]. Zhou B, Hui SC, Fong ACM (2006) Efficient sequential access pattern mining for Web recommendations. KES J 10:155–168

[11]. Yunhong Zhou, Dennis Wilkinson, Robert Schreiber and Rong Pan. Large-scale Parallel Collaborative Filtering for the Netflix Prize.

[12]. Wikipedia. https://en.wikipedia.org/wiki/Levenshtein_distance.

[13]. Wikipedia. https://en.wikipedia.org/wiki/Collaborative_filtering