**Article Title**

A standardized dataset for recommendation applications

**Authors**

Loc Nguyen1, Ali A. Amer2

**Affiliations**

1 Independent scholar, Vietnam

2 TAIZ University, Yemen

**Corresponding authors**

Loc Nguyen (ngphloc@yahoo.com)

Ali A. Amer (ali.amer@zu.ac.ae)

**Abstract**

Rating dataset for recommendation applications is not complicated and thus, storing it is not difficult. However, the complex degree will be increased if recommendation applications need much more additional information such as customer profiles, item profiles, and context information which go beyond common rating information. Therefore, we propose the Hudup dataset which aims to achieve two purposes: 1. Hudup dataset is organized in standardized structure in order to simplify the complex of extended rating dataset. 2. Programming objects in abstract level will provide researchers facilities to access and process dataset. We also make experiments on Hudup dataset by testing similarity measures built in nearest neighbors (NN) algorithm which is a collaborative filtering approach in recommendation research. Experimental design with splitting dataset and calculating sparse ratio become easy and simple with standardized Hudup dataset.

**Keywords**

Rating dataset, rating matrix, recommendation system, collaborative filtering, nearest neighbors algorithm, similarity measures.

**Specifications table**

|  |  |
| --- | --- |
| Subject |  |
| Specific subject area |  |
| Type of data |  |
| How data were acquired |  |
| Data format |  |
| Parameters for data collection |  |
| Description of data collection |  |
| Data source location |  |
| Data accessibility |  |
| Related research article |  |

**Value of the data**

In recommendation applications, rating information is simple, for instance, rating value is the degree of interest that a user assigns to an item. Rating information is modeled by a foursome (user identifier, item identifier, rating value, date to rate). Thus, it is easy to store collection of rating foursomes as rating matrix, which causes that it is not difficult to model and store rating dataset. However, a good rating dataset needs more additional information beyond rating foursomes, such as user profiles, item profiles, and context information along with different storage systems such as file system, FTP system, and database management system (DBMS). In general, there are two requirements for a good rating dataset:

1. It requires the data standardization in which all rating information and additional data (user profiles, item profiles, context information) along with different data types are organized in logic and solid structure.
2. It requires to model rating dataset by programming objects. These objects provide users and programmers facilitated properties and methods to access and process the dataset.

Therefore, the rating dataset in this research called Hudup dataset is composed of ten units and is organized into two abstract levels and one physical level, which aims to satisfy such two requirements. Hudup dataset is described in the section of data description. There are many collected rating datasets which are popular in research communities, for example, Movielens, Film Trust, Book Crossing, and Jester Joke. These datasets are raw datasets which will be converted into Hudup dataset before making experiments on them. In other words, Hudup framework will convert other datasets into Hudup dataset and then experiments are made on Hudup dataset instead raw datasets. Another strong point of Hudup dataset is its simple structure. Indeed, system administrators can modify easily and directly Hudup dataset without support of Hudup framework.

**Data description**

The structure of Hudup dataset has 10 main units such as “*hdp\_config*”, “*hdp\_account*”, “*hdp\_attribute\_map*”, “*hdp\_nominal*”, “*hdp\_user*”, “*hdp\_item*”, “*hdp\_rating*”, “*hdp\_context\_template*”, “*hdp\_context*”, and “*hdp\_sample*”. In other words, Hudup dataset is composed of such ten units. For each unit name, the prefix “hdp” is abbreviation of Hudup. Each unit owns attributes known as properties or fields. Unit “hdp\_config” establishes basic configurations over entire dataset in form of key-value pairs. Unit “hdp\_account” contains user-access information including account name (username), password, and privileges of user. Hudup dataset can interact with external databases via mapping mechanism. For instance, unit “hdp\_attribute\_map” maps an attribute of a special unit to a field of an external database table. Mapping mechanism improves adaptability of Hudup framework. When processing many data types, we often cope with nominal type which is names. Unit “hdp\_nominal” helps us to store such nominals or names as integers, which facilitate data processing. Units “hdp\_user” and “hdp\_item” contain information about user and item. Please distinguish user from account here. In recommendation, users here as customers and items are goods. Attributes (fields) of “hdp\_user” and “hdp\_item” are not fixed, which are dependent on applications. Unit “hdp\_rating” contains ratings of users (customers) on items. It includes four important attributes such as user identifier, item identifier, rating value, and rating date. In recommendation context, when a user gives rating on an item, there is context information connected with the rating event. For example, customers often go shopping on Saturday and so Saturday is context information. Units “hdp\_context\_template” and “hdp\_context” are used to model context-aware recommendation. Context stored in unit “hdp\_context” can be categorized into three main types in order to answer three questions “when, where and who” as follows. Time type indicates the time when user makes a purchase, for example: date, day of week, month, year. Location type indicates the place where user makes a purchase, for example: shop, market, theater, coffee house. Companion type indicates the persons with whom user makes a purchase, for example: alone, friends, girlfriend/boyfriend, family, co-workers. Actually, these context types are stored in unit “hdp\_context\_template”. The last unit is “sample” which is stored any information different from recommendation applications. Actually, unit “sample” is similar to common tables in databases; especially, in statistical applications, it contains sample information. It allows Hudup framework to cover more applications beyond recommendation process. In general, Hudup dataset and its units have the top-most abstract level, which are realized into the lower abstract level with programming objects which are in turn stored physically as database table, CSV files, Excel files, etc. So, Hudup dataset has two abstract levels and one physical level as follows:

Diagram

Description automatically generated

These levels will be described in detail later.

Recall that Hudup dataset is indeed an abstract object, which is instantiated into two forms such as programming object and physical storage. As a programming object, Hudup dataset is modeled as *Dataset* which is retrieved and accessed by programmers. As physical storage, Hudup dataset is stored as directory in file system or database in database management system (DBMS). Of course, there is always interaction between *Dataset* and physical storage, which is dependent on applications and purposes of Hudup framework. Thus, it is possible to identify Hudup dataset with the programming object *Dataset*. For programming language Java, *Dataset* is an interface which is implemented by realized objects. Similarly, each unit is an abstract object too, which is also instantiated into two forms such as programming object and physical storage. Regarding physical storage, unit is stored as CSV file, Excel file, or table for DBMS. However, programming objects for units are more plentiful. For example, “hdp\_config” is modeled as a map or dictionary including key-value pairs whereas “hdp\_attribute\_map” is modeled by both objects *InternalRecord* and *ExternalRecord*. Units “hdp\_account”, “hdp\_user”, and “hdp\_item” are modeled as *Profile* along with *Attribute*. Unit “hdp\_nominal” is modeled as *Nominal* object. Unit “hdp\_rating” is modeled as collections of *RatingVector*. The object *RatingVector* is collection of ratings which are given by a user. Units “hdp\_context\_template” and “hdp\_context” are modeled as *ContextTemplate* and *Contex*t, respectively. Recall that objects *Dataset*, *InternalRecord*, *ExternalRecord*, *Profile*, *Attribute*, *Nominal*, *RatingVector*, *ContextTemplate*, and *Context* are programming objects which provide properties and facilitated methods for programmers to process and access units. Obviously, each unit can be accessed by some specified objects in powerful manner, but *Profile* object can model all objects. In other words, all units can be accessed by *Profile* object which is the most flexible object. Please refer to the book chapter “Hudup: A Framework of E-commercial Recommendation Algorithms” (Nguyen & Do, 2015) to know some of these objects. Of course, *Dataset* has many methods to retrieve all other objects. In general, these objects are also abstract. However, Hudup dataset and units has the top-most abstract level.

In physical storage system, units are stored as CSV files, Excel files or database tables in form of tables whose columns are fields or attributes. Programming objects provides properties and method to access and process these files and tables. Following table lists fields for each unit.

|  |  |
| --- | --- |
| **Table** | **Fields** |
| “hdp\_config” | attribute, attribute\_value |
| “hdp\_account” | account\_name, account\_password, account\_privs |
| “hdp\_attribute\_map” | internal\_unit, internal\_attribute\_name, internal\_attribute\_value,  external\_unit, external\_attribute\_name, external\_attribute\_value |
| “hdp\_nominal” | nominal\_ref\_unit, attribute, nominal\_index, nominal\_value, nominal\_parent\_index |
| “hdp\_user” | userid, user\_type, field1, field2, etc. |
| “hdp\_item” | itemid, item\_type, field1, field2, etc. |
| “hdp\_rating” | userid, itemid, rating, rating\_date |
| “hdp\_context\_template” | ctx\_templateid, ctx\_name, ctx\_type, ctx\_parent |
| “hdp\_context” | userid, itemid, ctx\_templateid, ctx\_value, rating\_date |
| “hdp\_sample” | sample\_field1, sample\_field2, sample\_field3, etc. |

Thus, unit is called table in storage system. Following is description of table “hdp\_config” which stores configuration information.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Data  type | Is  key | Allow  null | Description |
| attribute | Text | Yes | No | Name of configured property. |
| attribute\_value | Text | No | No | Value of configured property. |

Following is description of table “hdp\_account” which stores account information.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Data  type | Is  key | Allow  null | Description |
| account\_name | Text | Yes | No | Account name also known as username. |
| account\_password | Encrypted  text | No | No | Account password. |
| account\_privs | Integer | No | No | An integer specifying account privileges. |

Following is description of table “hdp\_attribute\_map” which maps an attribute of an internal unit to a field of an external database table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Data  type | Is  key | Allow  null | Description |
| internal\_unit | String | Yes | No | Name of internal unit. |
| internal\_attribute\_name | String | Yes | No | Name of attribute of such internal unit. |
| internal\_attribute\_value | String | Yes | No | Value of such internal attribute. |
| external\_unit | String | No | No | Name of external table (or CSV file, Excel file). |
| external\_attribute\_name | String | No | No | Name of field of such external table. |
| external\_attribute\_value | String | No | No | Value of such external field. |

Following is description of table “hdp\_nominal” which stores nominals or names as integers.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Data  type | Is  key | Allow  null | Description |
| nominal\_ref\_unit | String | Yes | No | Name of unit which contains the attribute whose value is nominal. |
| attribute | String | Yes | No | Name of the attribute whose value is nominal. |
| nominal\_index | Integer | Yes | No | Index of the nominal. |
| nominal\_value | String | No | No | Nominal (value in text) of the attribute. |
| nominal\_parent\_index | Integer | No | Yes | Parent index of the nominal. |

As a result, the unit (specified by “nominal\_ref\_unit”) which contains the attribute (specified by “attribute”) whose nominal specified by “nominal\_value” stores the integer “nominal\_index” instead of storing the text “nominal\_value”. Nominals can be structured in hierarchy, which is modeled by the attribute “nominal\_parent\_index”.

Followings are descriptions of tables “hdp\_user” and “hdp\_item” which stores user information and item information. In recommendation, users here as customers and items are goods. Fields (attributes) of “hdp\_user” and “hdp\_item” are not fixed, which are dependent on applications. For instance, there can be field3, field4,…, field*n* in “hdp\_user” and “hdp\_item”.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Data  type | Is  key | Allow  null | Description |
| userid | Integer | Yes | No | User (customer) identifier. |
| user\_type | Integer | No | No | User type. |
| field1 | Any type | No | Yes | Field 1. |
| field2 | Any type | No | Yes | Field 2. |
| … | Any type | No | Yes | … |
| field*n* | Any type | No | Yes | Field *n*. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Data  type | Is  key | Allow  null | Description |
| itemid | Integer | Yes | No | Item identifier. |
| item\_type | Integer | No | No | Item type. |
| field1 | Any type | No | Yes | Field 1. |
| field2 | Any type | No | Yes | Field 2. |
| … | Any type | No | Yes | … |
| field*n* | Any type | No | Yes | Field *n*. |

Following is description of table “hdp\_rating” which contains ratings of users (customers) on items.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Data  type | Is  key | Allow  null | Description |
| userid | Integer | Yes | No | User (customer) identifier which points to “userid” in table “hdp\_user”. |
| itemid | Integer | Yes | No | Item identifier which points to “itemid” in table “hdp\_item”. |
| rating | Real | No | No | Rating value that the user specified by “userid” gave on the item specified by “itemid”. |
| rating\_date | Date | Yes | No | Rating date that the user specified by “userid” rated on the item specified by “itemid”. |

Following is description of table “hdp\_context\_template” which store context templates known as context types (time, location, accompany). In current version, context templates are structured in hierarchy specified by the attribute “ctx\_parent”.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Data  type | Is  key | Allow  null | Description |
| ctx\_templateid | Integer | Yes | No | Context template identifier. |
| ctx\_name | String | No | No | Context name. |
| ctx\_type | Integer | No | No | Context type (time, location, accompany). Actually, it is encrypted as integer. |
| ctx\_parent | Integer | No | Yes | Identifier of parent context template of current template, which points to another “ctx\_templateid”. |

Every context template describes a context itself as data type. In recommendation applications, the event that a customer (user) rates on an item in a concrete context *c* implies that the context template named *C* (“ctx\_name” = *C*) is assigned with a concrete value *c*. Therefore, context-aware rating information is stored in table “hdp\_context” as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Data  type | Is  key | Allow  null | Description |
| userid | Integer | Yes | No | User identifier (customer identifier) which points to “userid” in table “hdp\_rating”. |
| itemid | Integer | Yes | No | Item identifier which points to “itemid” in table “hdp\_rating”. |
| ctx\_templateid | Integer | Yes | No | Context template identifier which points to “ctx\_templateid” in table “hdp\_context\_template”. |
| ctx\_value | Integer | No | No | Value of context template identifier. |
| rating\_date | Date | Yes | No | Rating date which points to “rating\_date” in table “hdp\_rating”. |

The table “hdp\_context” is always associated with the table “hdp\_rating”, which can be considered as the second “hdp\_rating” because it stores context information when the user specified by “userid” rates on the item specified by “itemid” instead of storing rating value (a real number) as the table “hdp\_rating” does.

Following is description of table “hdp\_sample” which is normal table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Data  type | Is  key | Allow  null | Description |
| sample\_field1 | Any type |  |  | Field 1. |
| sample\_field2 | Any type |  |  | Field 2. |
| sample\_field3 | Any type |  |  | Field 3. |
| … | Any type |  |  | … |
| sample\_field*n* | Any type |  |  | Field *n*. |

Now ten main tables corresponding to ten main units are described. Recall that, in the table “hdp\_context”, a context is composed of a context template and a value of such template. In other words, a context is an instance of a context template. Every context template specified by “ctx\_templateid” can have one or more values. Therefore, for each context template *k* (“ctx\_templateid” = *k*) there is an extra table named “*hdp\_context\_template\_k\_profile*” which is indeed profile of context template *k*. Following is description of table “hdp\_context\_template\_k\_profile”. Of course, there are many such tables.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Data  type | Is  key | Allow  null | Description |
| ctx\_value | Integer | Yes | No | A value of context template *k*. |
| field1 | Any type | No | Yes | Field 1. |
| field2 | Any type | No | Yes | Field 2. |
| … | Any type | No | Yes | … |
| field*n* | Any type | No | Yes | Field *n*. |

Of course, custom fields (field1, field2,…, field*n*) of table “hdp\_context\_template\_k\_profile” are not fixed, which is dependent on applications. Note, table “hdp\_context\_template\_k\_profile” is not important because it can be inexistent. However, if it is inexistent, we do not know additional information about values of context template.

**Experimental design, materials, and methods**

In this research, Hudup dataset is used to test similarity measures built in nearest neighbors (NN) algorithm with regard to collaborative filtering in recommendation applications. Note, NN algorithm uses similarity measures as essential manner to support collaborative filtering. The testing process is divided into two sub-processes such as estimation sub-process and recommendation sub-process with metrics such as absolute average error (MAE), precision, and recall. Here we focus on how to make experimental design on Hudup dataset. In this research, the raw dataset (base dataset) of Hudup dataset is Movielens.

The problem in recommendation is how to determine the number of recommended items denoted *C* which is the length of recommended vector. As a convention, *C* is called recommendation count. The count *C* cannot be too small or too large. If it is too small, evaluation is inaccurate. Otherwise, if it is too large, evaluation task will run slowly. Some researches set fixed number whereas other researches changed such number over some values such as 10, 20, and 100. We proposed a method to determine *C* based on dataset with purpose that *N* will be more accurate and objective. The proposed method is dynamic and takes advantages of a so-called sparse-relevant ratio. This ratio is the ratio of the count of relevant ratings to the count of cells with note that the count of cells is product of user number and item number, which is size of rating matrix. Recall that a relevant rating is larger than average rating and the count of cells is sum of the count of rating values and the count of missing values. Following equation specifies sparse-relevant ratio denoted *sr*.

*sr* = the-count-of-relevant-ratings / (|***U***| \* |***V***|)

Note, |***U***| is the number of users and |***V***| is the number of items. We calculate recommendation count *C* dynamically according to both dataset and each rating vector *ui*. Let *C*(*ui*) be the recommendation count for user *i*, which means that NN algorithms will recommend at least *C*(*ui*) items to user *i*. Following equation specifies *C*(*ui*).

Where *T* is the number of items with note that every item included in *T* is rated by at least one user. Of course, *T* is smaller than or equal to the number of users |***U***|. Note, |*Ii*| is the number of items rated by user *i*. The quantity |*Ii*| is not redundant because real recommendation systems always recommend a user items that she/he do not either know or rate yet. If |*Ii*| is too much smaller than *T* (|*Ii*| << T), *C*(*ui*) can be calculated as follows:

Recall that in our experiments, Hudup dataset (Movielens dataset) is divided into 5 folders and each folder has one training set and one testing set. For each folder, *T* and sparse-relevant ratio *sr* are calculated on training set but |*Ii*| is determined on testing set, of course. For example, suppose one among 5 folders divided from Movielens has training set *d*1 and testing set *t*1. The number of users in *d*1 is 943 and the number of items in *d*1 is 1,584. Because every item in *d*1 is rated by at least one user, we have *T* = 1,584. Training set *d*1 has 50,000 rating values but only 27,712 rating values are relevant. So sparse-relevant ratio is *sr* = 27,712 / (943\*1584) ≈ 1.86%. Suppose it is necessary to make recommendation on user rating vector *u*12 (in testing set *t*1) which has 23 rating values. Hence, recommendation count for user 12 is *C*(*u*12) = 1.86% \* (1,584 – 23) ≈ 29.

**Acknowledgements**

**Competing interests**

The authors declare that they have no known competing financial interests or personal relationships which have, or could be perceived to have, influenced the work reported in this article.

**References**

Nguyen, L., & Do, P. (2015). Hudup: A Framework of E-commercial Recommendation Algorithms. In A. Fred, J. Dietz, D. Aveiro, K. Liu, & J. Filipe (Ed.), *Final Program and Book of Abstracts of The 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2015)* (p. 56). Lisbon: SCITEPRESS. doi:10.13140/RG.2.2.27533.84969/1

Ricci, F., Rokach, L., Shapira, B., & Kantor, P. B. (2011). *Recommender Systems Handbook* (Vol. I). (F. Ricci, L. Rokach, B. Shapira, & P. B. Kantor, Eds.) Springer New York Dordrecht Heidelberg London.