**A Framework of E-commercial Recommendation Algorithms**

Loc Nguyen

Sunflower Soft Company, Ho Chi Minh city, Vietnam

**Abstract**

Recommendation algorithm is very important for e-commercial websites when it can recommend online customers favorite products, which results out an increase in sale revenue. I propose the framework of e-commercial recommendation algorithms. This is a middleware framework or “operating system” for e-commercial recommendation system, which support scientists and software developers build up their own recommendation algorithms based this framework with low cost, high achievement and fast speed.

**Keywords:** recommendation algorithm, recommendation server, middleware framework.

**1. Introduction**

The product is the “Framework of e-commercial recommendation solutions”, named **Hudup**. This is a middleware framework or “operating system” for e-commercial recommendation software, which support scientists and software developers build up their own recommendation solutions based this framework. The term “recommendation solutions” mentions the computer algorithms which aim to recommend online customers an introduction list of items such as books, products, services, news papers, fashion clothes and etc on commercial websites with expectation that customers will like these recommended items. The goal of recommendation algorithms is to gain high sale revenue.

You need to develop a recommendation solution for online-sale website. You, a scientist, invent a new algorithm after researching many years. Your solution is excellent and very useful and so you are very exciting but:

1. You cope with complicated computations when analyzing big data and there are a variety of heterogeneous in recommendation studies.
2. It is impossible for you to evaluate your algorithm according to standard metrics.
3. There is no simulation environment or simulator for you to test the feasibility of your algorithm.

The innovative product “Framework of e-commercial recommendation solutions” supports you to solve perfectly three above difficulties and so following are your achievements:

1. Realizing your solution is very fast and easy.
2. Evaluating your solution according to standard metrics by the best way.
3. Determining the feasibility of your algorithm in real-time applications.

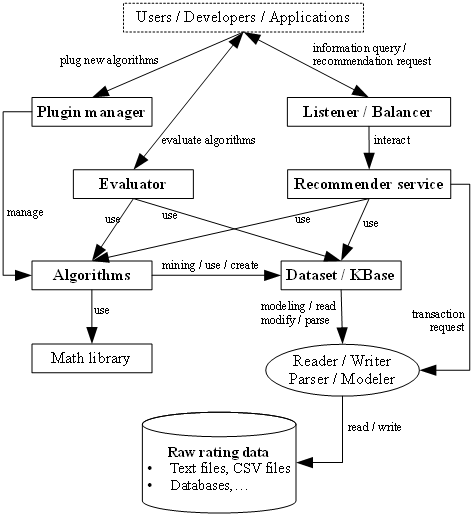
The product has another preeminent function which is to provide two optimized algorithms so that it is very convenient for you to assess and compare different solutions. The product aims to help you, a scientist or software developer, to solve three above core problems. The product proposes three solution stages for developing a recommendation algorithm.

1. *Base stage*: build up algorithm model and data model to help you to create new software with lowest cost.
2. *Evaluation stage*: build up evaluation metrics and algorithm evaluator to help you to assess your own algorithm.
3. *Simulation stage*: build up recommendation server or simulator help you to test the feasibility of your algorithm.

There are now some other open source software similar to my product; the reference is the brief list of them [1] – [14]. After surveying 14 other typical products, my product is the unique and most optimal if the function to support scientists and software developers through 3 stages such as algorithm implementation, quality assessment and experiment is considered most. Moreover the architecture of product is very flexible and highly customizable. Evaluation metrics to qualify algorithms are standardized according to pre-defined templates so that it is possible for software developer to modify existing metrics and add new metrics.

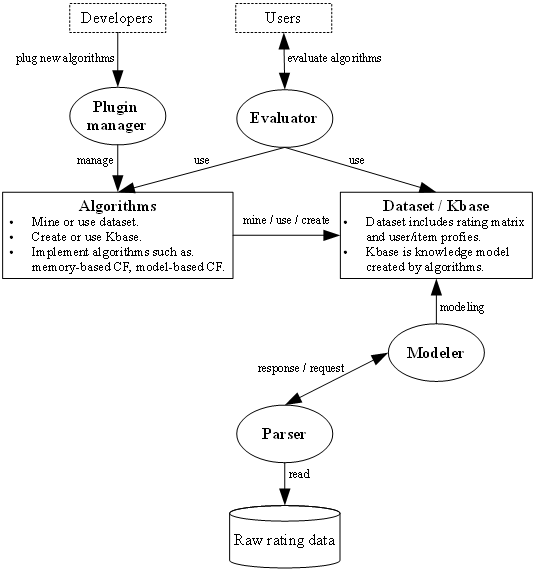
**2. Description of Product**

The product is computer software composed of three main modules such as *Plug-in manager*, *Evaluator* and *Recommender Service*, which correspond to solution stages such as base stage, evaluation stage and simulation stage. Figure 1 depicts the general architecture of product.



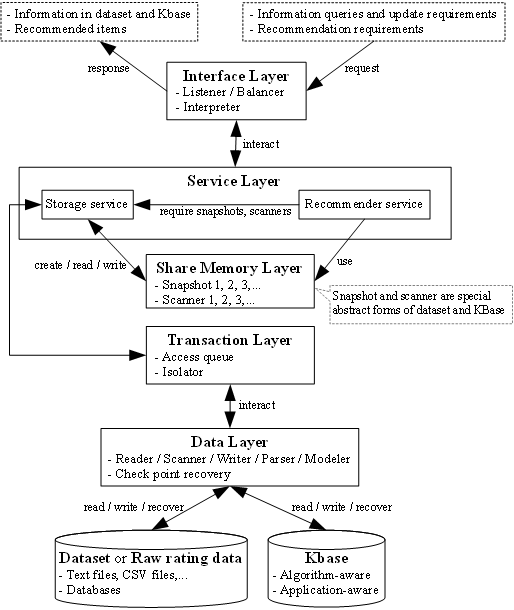
**Figure 1:** General architecture of product

Figure 2 depicts the sub-architecture of Evaluator.



**Figure 2:** Architecture of Evaluator

Figure 3 depicts the sub-architecture of Recommender Service.

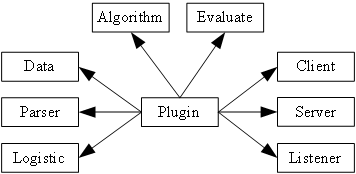


**Figure 3:** Architecture of Recommender Service or Recommender Server

The general architecture is decomposed into 9 packages as follows:

1. *Data* package: standardizing and modeling data in abstract level.
2. *Parser* package: analyzing and processing data.
3. *Algorithm* package: modeling recommendation algorithm in abstract level.
4. *Evaluate* package implements built-in evaluation mechanism of the framework. It also establishes common evaluation metrics. *Evaluate* package supports mainly module *Evaluator*.
5. *Client* package, *Server* package and *Listener* package: providing recommender service in client-server network.
6. *Logistic* package: providing computational and mathematic utilities.
7. *Plug-in* package: managing algorithms and evaluation metrics.

Three main modules *Plug-in manager*, *Evaluator* and *Recommender Service* are constituted of such 9 packages. Figure 4 depicts such nine packages of product.



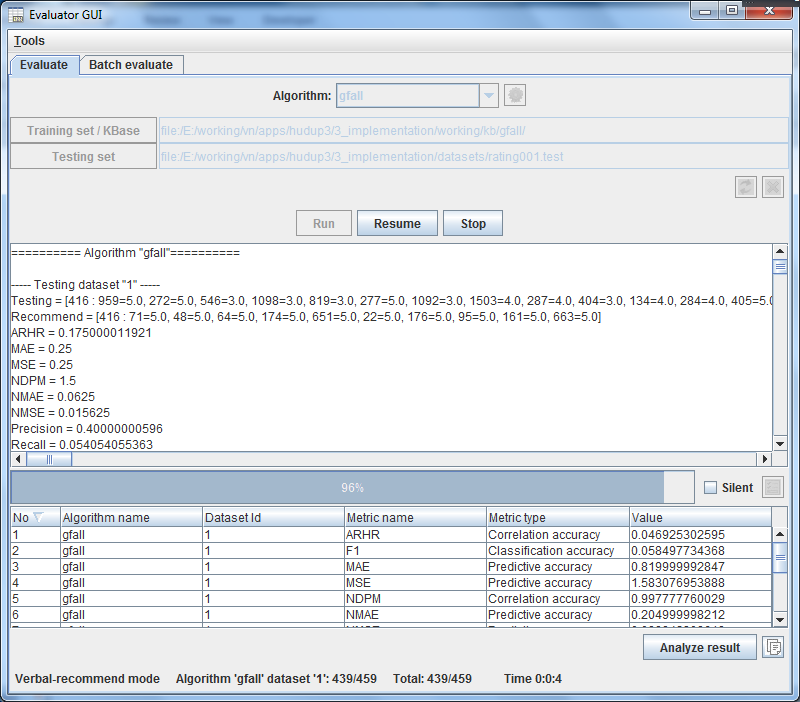
**Figure 4:** Nine packages of product

Each package includes many software classes constituting internal class diagrams. Especially, the *Algorithm* package provides two optimized algorithms such as collaborative filtering algorithm based on mining frequent itemsets and collaborative filtering algorithm based on Bayesian network inference.

The product helps you to build up a recommendation algorithm fast and easily. Moreover, it is very convenient for you to assess the quality and feasibility of your own algorithm in real-time application. Suppose you want to set up a new collaborative filtering algorithm so-called Green Fall, instead of writing big software with a huge of complicated tasks such as processing data, implementing algorithm, implementing evaluation metrics, testing algorithm, creating simulation environment and etc; what you need to do is to follow three below steps:

1. Inheriting *Recommender* class in *Algorithm* package and hence, implementing your idea in two methods *estimate* and *recommend* of two these classes.
2. Starting up the module *Evaluator*, please see figure 1 and 2, so as to evaluate and compare Green Fall with other algorithms via pre-defined evaluation metrics.
3. Configuring *Recommender service* in order to embed Green Fall into such service. After that starting up *Recommender service* so as to test the feasibility of Green Fall in real-time applications.

Operations in three above steps are very simple; there are mainly configurations via software graphic user interface (GUI), except that you require setting up your idea by programming code lines in step 1. Because algorithm model is designed and implemented very strictly, what you program is encapsulated in two methods *estimate* and *recommend* of *Recommender* class. The average time cost to build up and test an algorithm is around 2 years but it remains 1 week for you to realize your idea if you use my product. It means that the algorithm development cost decreased very much and so it only takes 1% origin expenditure. It is really exciting work. Figure 5 shows a screen shot of module *Evaluator*.



**Figure 5:** Screen shot of module Evaluator

The product home link is:

“<https://sites.google.com/site/ngphloc/st/products/hudup>”

**Acknowledgements**

This product is the place to acknowledge Ms. Do, Phung T. M. – University of Information Technology, Vietnam National University and Mr. Vu, Dong N. who gave me valuable comments and advices. These comments help me to improve this product.

**References**

1. Brožovský, L. (2006, August). ColFi - Recommender System for a Dating Service. (V. Petříček, Ed.) Prague, Czech Republic: Charles University in Prague.
2. Caraciolo, M., Melo, B., & Caspirro, R. (2011). Crab - Recommender systems in Python. Muriçoca Labs.
3. Chen, T., Zhang, W., Lu, Q., Chen, K., Zheng, Z., Yu, Y., et al. (2012). SVDFeature: A Toolkit for Feature-based Collaborative Filtering. 1.2.2. China: APEX Data & Knowledge Management Lab.
4. Dato-Team. (2013, October 15). GraphLab Create™. (C. Guestrin, Ed.) Seattle, Washington, USA: Dato, Inc.
5. Ekstrand, M., Kluver, D., He, L., Kolb, J., Ludwig, M., & He, Y. (2013). LensKit - Open-Source Tools for Recommender Systems. GroupLens Research, University of Minnesota.
6. Gantner, Z., Rendle, S., Drumond, L., & Freudenthaler, C. (2013). MyMediaLite Recommender System Library. 3.10. University of Hildesheim and The European Commission 7th Framework Programme.
7. Hahsler, M. (2014, 08 18). recommenderlab: Lab for Developing and Testing Recommender Algorithms. 0.1-5. NSF Industry/University Cooperative Research Center for Net-Centric Software & Systems.
8. Lemire, D. (2003). Cofi: A Java-Based Collaborative Filtering Library. Canada: National Research Council of Canada.
9. Lew, D., & Sowell, B. (2007). Carleton Recommender systems. (D. Musicant, Ed.) Northfield, Minnesota, USA: Carleton College.
10. Mahout-Team. (2013). Apache Mahout™. The Apache Software Foundation.
11. Microsoft, MyMediaPC, RichHanbidge, MyMediaWp2Lead, MyMediaWp3Lead, MyMediaWp4Lead, et al. (2013). MyMedia Dynamic Personalization and Recommendation Software Framework Toolkit. MyMedia project funded through the EU Framework 7 Programme Networked Media initiative.
12. Smart-Agent-Technologies. (2013). easyrec. Research Studios Austria Forschungsgesellschaft mbH.
13. Telematica-Instituut. (2007, June 29). Duine Framework. Telematica Instituut/Novay.
14. Vogoo-Team, & DROUX, S. (2008). Vogoo PHP LIB. 2.2. SourceForge.