

On both Cold-Start and Long-Tail Recommendation with Social Data

Submitted in partial fulfillment of the
requirements for the award of
Bachelor of Science degree in Computer Science

By
R.Mathivathani(39290064)
J.Lavanya(39290057)



**DEPARTMENT OF COMPUTER SCIENCE
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Accredited with Grade "A" by NAAC

**JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI
- 600 119**

MARCH - 2022



SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **R.MATHIVATHANI (39290064)** and **J.LAVANYA(39290057)** who carried out the project entitled **ON BOTH COLD - START AND LONG - TAIL RECOMMENDATION WITH SOCIAL DATA** under my supervision

Internal Guide

Dr. M. MAHESHWARI

Head of the Department

Dr. L. LAKSHMANAN,M.E,P.H.D

Internal Examiner

**External
Examiner**

DECLARATION

R.MATHIVATHANI (39290064) hereby declare that the Project Report entitled **ON BOTH COLD - START AND LONG - TAIL**

RECOMMENDATION WITH SOCIAL DATA done by me under the guidance of **Dr. M. MAHESHWARI** is submitted in partial fulfillment of the requirements for the award of Bachelor of Science degree in **COMPUTER SCIENCE.**

DATE:

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T.Sasikala M.E., Ph.D, Dean**, School of Computing **Dr. L. Lakshmanan M.E., Ph.D. ,** and **Dr.S.Vigneshwari M.E., Ph.D. Heads** of the Department of Computer Science and Engineering for providing necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide Dr. M. MAHESHWARI for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

The number of “hits” has been widely regarded as the lifeblood of many web systems, e.g., e-commerce systems, advertising systems and multimedia consumption systems. However, users would not hit an item if they cannot see it, or they are not interested in the item. Recommender system plays a critical role of discovering interesting items from near-infinite inventory and exhibiting them to potential users. Yet, two issues are crippling the recommender systems. One is “how to handle new users”, and the other is “how to surprise users”. The former is well-known as cold-start recommendation. In this paper, we show that the latter can be investigated as long-tail recommendation. We also exploit the benefits of jointly challenging both cold-start and long-tail recommendation, and propose a novel approach which can simultaneously handle both of them in a unified objective. For the cold-start problem, we learn from side information, e.g., user attributes, user social relationships, etc. Then, we transfer the learned knowledge to new users. For the long-tail recommendation, we decompose the overall interesting items into two parts: a low-rank part for short-head items and a sparse part for long-tail items. The two parts are independently revealed in the training stage, and transferred into the final recommendation for new users. Furthermore, we effectively formulate the two problems into a unified objective and present an iterative optimization algorithm. A fast extension of the method is proposed to reduce the complexity, and extensive theoretical analysis are provided to proof the bounds of our approach. At last, experiments of social recommendation on various real-world datasets

TABLE OF CONTENTS

CHAPTER No	TITLE	PAGE
	LIST OF FIGURES	Vi
	LIST OF ABBREVIATIONS	Vii
1	INTRODUCTION	
1.1	About the Project	
2	SYSTEM ANALYSIS	
2.1	Existing system	
2.2	Proposed system	
3	REQUIREMENTS SPECIFICATION	
3.1	Introduction	
3.2	Hardware and Software specification	
3.3	Technologies Used	
3.3.1	Java	
3.3.3.1	Introduction to Java	
3.3.1.2	Working of Java	
4	Project Purpose and Scope	
4.1	Purpose	
4.2	Project Scope	

4.3	Product Perspective	
4.4	System Features	
4.5	Design and Implementation Constraints	
4.6	Other Nonfunctional Requirements	
5	SYSTEM DESIGN	
5.1	Architecture Diagram	
6	SYSTEM DESIGN – DETAILED	
6.1	Modules	
6.2	Module explanation	
7	CODING AND TESTING	
7.1	Coding	
7.2	Coding standards	
7.3	Test procedure	
7.4	Test data and output	
	SOURCE CODE	
	SNAP SHOTS	
	REFERENCES	

LIST OF FIGURES

5.1 System Design

5.2 Sequence Diagram

5.3 Use Case Diagram

5.4 Activity Diagram

5.5 Collaboration Diagram:

5.6 Data Flow Diagram

5.7 Class Diagram

LIST OF ABBREVIATIONS

JDK	Java Development Toolkit
DEX	Dalvik Executables
TCP	Transmission Control Protocol
IP	Internet Protocol
HTTP	Hyper Text Transfer Protocol
ADT	Android Development Tool

CHAPTER 1

INTRODUCTION

Aim:

The main aim of this project is to achieve the cold start and long tail problems and recommend the products using social data.

Synopsis:

The number of “hits” has been widely regarded as the lifeblood of many web systems, e.g., e-commerce systems, advertising systems and multimedia consumption systems. However, users would not hit an item if they cannot see it, or they are not interested in the item. Recommender system plays a critical role of discovering interesting items from near-infinite inventory and exhibiting them to potential users. Yet, two issues are crippling the recommender systems. One is “how to handle new users”, and the other is “how to surprise users”. The former is well-known as cold-start recommendation. In this paper, we show that the latter can be investigated as long-tail recommendation. We also exploit the benefits of jointly challenging both cold-start and long-tail recommendation, and propose a novel approach which can simultaneously handle both of them in a unified objective. For the cold-start problem, we learn from side information, e.g., user attributes, user social relationships, etc. Then, we transfer the learned knowledge to new users. For the long-tail recommendation, we decompose the overall interesting items into two parts: a low-rank part for short-head items and a sparse part for long-tail items. The two parts are independently revealed in the training stage, and transferred into the final recommendation for new users. Furthermore, we effectively formulate the two problems into a unified objective and present an iterative optimization algorithm. A fast extension of the method is proposed to reduce the complexity, and extensive theoretical analysis are provided to proof the bounds of our approach. At last, experiments of social recommendation on various real-world datasets, e.g., images, blogs, videos and musics, verify the superiority of our approach compared with the state-of-the-art work.

CHAPTER 2

SYSTEM ANALYSIS

2.1 EXISTING SYSTEM

Recommender system plays a critical role of discovering interesting items from near-infinite inventory and exhibiting them to potential users. Yet, two issues are crippling the recommender systems. One is “how to handle new users”, and the other is “how to surprise users”. The former is well-known as cold-start recommendation. In this paper, we show that the latter can be investigated as long-tail recommendation. Cold-start problem has been regarded as a quite challenging problem in the community and no direct information available. In existing system have revealed that recommending popular items do not bring much benefits to either users or content providers.

2.2 PROPOSED SYSTEM

Based on the problems due to cold start and long tail, products in the ecommerce retailers getting sold out so soon . And some kind of products remains stagnant for a long duration. Thus to overcome these problems we are going to track the selling of products as well as recommending the products to the customers of same kind based on the previous purchases by using clustering and classification we achieve this First user register the details like name ,password,address and registers then login with valid credentials then the large amount of transactional data collected from customer for segmentation and classification and these data are

preprocessed and cluster and classify the data cold product classified based on sale and long tail is classified based product that getting sold frequently after that cold products that are available in the ecommerce site will be recommended to the user to make retailer some profit for the product and finally long tail problem solved by recommending relevant item and product that are not sold for long period gets attractive offer based on the user request. We use hadoop environment to do manipulation with the products dataset.

CHAPTER 3

REQUIREMENT SPECIFICATIONS

3.1 INTRODUCTION

“The success of any kind of social epidemic is heavily dependent on the involvement of people with a particular and rare set of social gifts,” writes Malcolm Gladwell in his bestseller *The Tipping Point* [1]. He further named these people as connectors, mavens and salesmen. The connectors know people, the mavens accumulate information of commodity, and the salesman persuade people to buy things. What’s particularly intriguing is that in online retail stores, recommender system [2], [3], [4], [5], [6] plays the three roles at the same time. Specifically, a recommender system discovers the interesting items from a near-infinite inventory and exhibits them to potential users. In other words, a recommender system must know people to find out potential users, must accumulate information to pick up interesting items, and must persuade the potential users to buy the recommended items. As a result, the success of on-line retailing is heavily dependent the recommender system. Actually, Amazon.com has received a 35% of their revenue from recommendations, and 75% of what people watch in Netflix is as a result of recommendations¹.

3.2 HARDWARE AND SOFTWARE SPECIFICATION

3.2.1 HARDWARE REQUIREMENTS

Hard Disk	:	250GB and Above
RAM	:	4GB and Above
Processor	:	I3 and Above(64 bit)
Wifi Router	:	1No.

3.2.2 SOFTWARE REQUIREMENTS

Windows 7

JDK 1.7

J2EE

Tomcat 6.0

MySQL

Hadoop

3.3 TECHNOLOGIES USED

- JAVA
- Android

3.3.1 JAVA

Java is an object-oriented programming language developed initially by James Gosling and colleagues at Sun Microsystems. The language, initially called Oak (named after the oak trees outside Gosling's office), was intended to replace C++, although the feature set better resembles that of Objective C.

3.3.1.1 INTRODUCTION TO JAVA

Java has been around since 1991, developed by a small team of Sun Microsystems developers in a project originally called the Green project. The intent of the project was to develop a platform-independent software technology that would be used in the consumer electronics industry. The language that the team created was originally called Oak.

The first implementation of Oak was in a PDA-type device called Star Seven (*7) that consisted of the Oak language, an operating system called GreenOS, a user interface, and hardware. The name *7 was derived from the telephone sequence that was used in the team's office and that was dialed in order to answer any ringing telephone from any other phone in the office.

Around the time the First Person project was floundering in consumer electronics, a new craze was gaining momentum in America; the craze was called "Web surfing." The World Wide Web, a name applied to the Internet's millions of linked HTML documents was suddenly becoming popular for use by the masses. The reason for this was the introduction of a graphical Web browser called Mosaic, developed by ncSA. The browser simplified Web browsing by combining text and graphics into a single interface to eliminate the need for users to learn many confusing UNIX and DOS commands. Navigating around the Web was much easier using Mosaic.

It has only been since 1994 that Oak technology has been applied to the Web. In 1994, two Sun developers created the first version of Hot Java, and then called Web Runner, which is a graphical browser for the Web that exists today. The browser was coded entirely in the Oak language, by this time called Java. Soon

after, the Java compiler was rewritten in the Java language from its original C code, thus proving that Java could be used effectively as an application language. Sun introduced Java in May 1995 at the Sun World 95 convention.

Web surfing has become an enormously popular practice among millions of computer users. Until Java, however, the content of information on the Internet has been a bland series of HTML documents. Web users are hungry for applications that are interactive, that users can execute no matter what hardware or software platform they are using, and that travel across heterogeneous networks and do not spread viruses to their computers. Java can create such applications.

3.3.1.2 WORKING OF JAVA

For those who are new to object-oriented programming, the concept of a class will be new to you. Simplistically, a class is the definition for a segment of code that can contain both data (called attributes) and functions (called methods).

When the interpreter executes a class, it looks for a particular method by the name of **main**, which will sound familiar to C programmers. The main method is passed as a parameter an array of strings (similar to the argv [] of C), and is declared as a static method.

To output text from the program, we execute the **println** method of **System.out**, which is java's output stream. UNIX users will appreciate the theory behind such a stream, as it is actually standard output. For those who are instead used to the Wintel platform, it will write the string passed to it to the user's program.

Java consists of two things :

- Programming language

- Platform

3.3.1.3 THE JAVA PROGRAMMING LANGUAGE

Java is a high-level programming language that is all of the following:

- Simple
- Object-oriented
- Distributed
- Interpreted
- Robust
- Secure
- Architecture-neutral
- Portable
- High-performance
- Multithreaded
- Dynamic

The code and can bring about changes whenever felt necessary. Some of the standard needed to achieve the above-mentioned objectives are as follows:

Java is unusual in that each Java program is both compiled and interpreted. With a compiler, you translate a Java program into an intermediate language called **Java byte codes** – the platform independent codes interpreted by the Java interpreter. With an interpreter, each Java byte code instruction is parsed and run on the computer. Compilation happens just once; interpretation occurs each time the program is executed. This figure illustrates how it works:

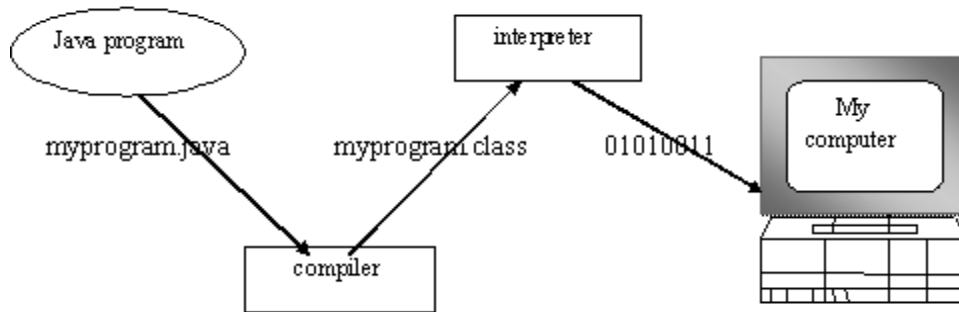
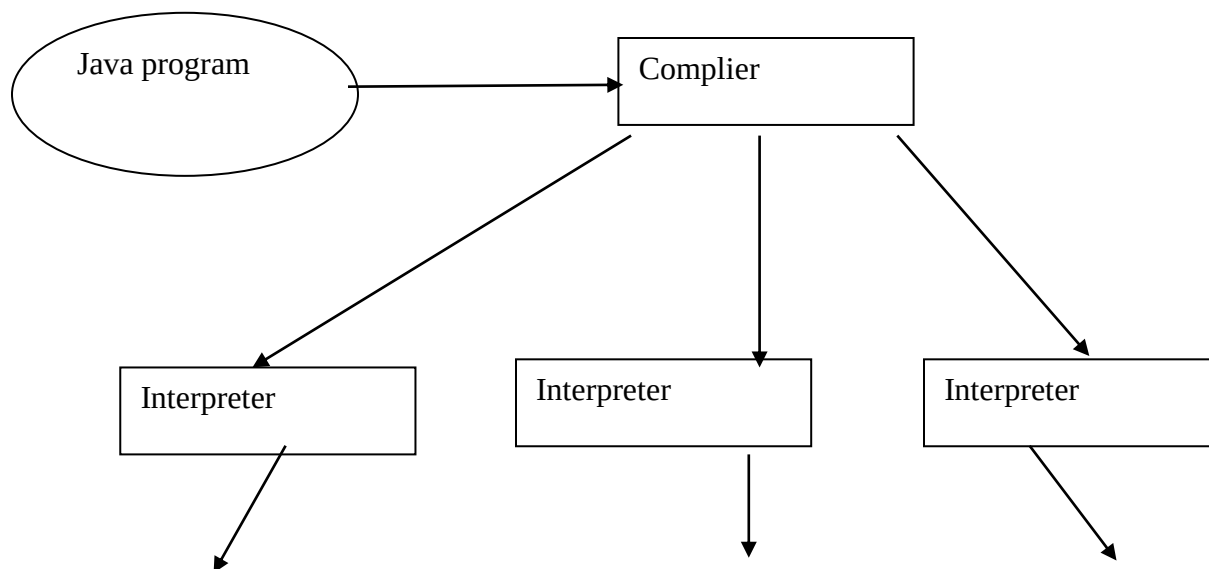


Fig.3.1

You can think of Java byte codes as the machine code instructions for the **Java Virtual Machine (JVM)**. Every Java interpreter, whether it's a Java development tool or a Web browser that can run Java applets, is an implementation of JVM. That JVM can also be implemented in hardware. Java byte codes help make “write once, run anywhere” possible.

You can compile your Java program into byte codes on any platform that has a Java compiler. The byte codes can then be run on any implementation of the JVM. For example, that same Java program can e run on Windows NT, Solaris and Macintos





PC-Compatible

Windows NT



Sun Ultra Solaris



Power macintosh

System 8

3.3.1.4 THE JAVA PLATFORM

A platform is the hardware or software environment in which a program runs. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other, hardware-based platforms. Most other platforms are described as a combination of hardware and operating system.

The Java platform has two components :

- The Java Virtual Machine (JVM)
- The Java Application Programming Interface (Java API)

You've already been introduced to the JVM. It's the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries (**packages**) of related components. The following figure depicts a Java program, such as an application or applet, that's running on the Java platform. As the figure shows, the Java API and Virtual Machine insulates the Java program from hardware dependencies.

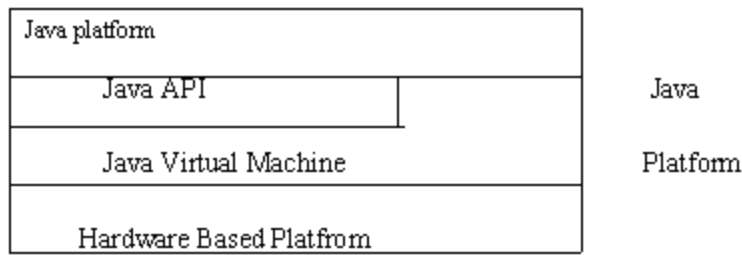


Fig.3.3

As a platform-independent environment, Java can be a bit slower than native code. However, smart compilers, weel-tuned interpreters, and just-in-time byte compilers can bring Java's performance close to that of native code without threatening portability.

3.3.2 APACHE TOMCAT SERVER

Apache Tomcat (formerly under the Apache Jakarta Project; Tomcat is now a top level project) is a web container developed at the Apache Software Foundation. Tomcat implements the servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems, providing an environment for Java code to run in cooperation with a web server. It adds tools for configuration and management but can also be configured by editing configuration files that are normally XML-formatted. Because Tomcat includes its own HTTP server internally, it is also considered a standalone web server.

Environment

Tomcat is a web server that supports servlets and JSPs. Tomcat comes with the Jasper compiler that compiles JSPs into servlets.

The Tomcat servlet engine is often used in combination with an Apache web server or other web servers. Tomcat can also function as an independent web server. Earlier in its development, the perception

existed that standalone Tomcat was only suitable for development environments and other environments with minimal requirements for speed and transaction handling. However, that perception no longer exists; Tomcat is increasingly used as a standalone web server in high-traffic, high-availability environments.

Since its developers wrote Tomcat in Java, it runs on any operating system that has a JVM.

Product features

Tomcat 3.x (initial release)

- implements the Servlet 2.2 and JSP 1.1 specifications
- servlet reloading
- basic HTTP functionality Tomcat 4.x
- implements the Servlet 2.3 and JSP 1.2 specifications
- servlet container redesigned as Catalina
- JSP engine redesigned as Jasper
- Coyote connector
- Java Management Extensions (JMX), JSP and Struts-based administration
- Tomcat 5.x
- implements the Servlet 2.4 and JSP 2.0 specifications
- reduced garbage collection, improved performance and scalability
- native Windows and Unix wrappers for platform integration
- faster JSP parsing

History

Tomcat started off as a servlet specification implementation by James Duncan Davidson, a software

architect at Sun. He later helped make the project open source and played a key role in its donation by Sun to the Apache Software Foundation.

Davidson had initially hoped that the project would become open-sourced and, since most open-source projects had O'Reilly books associated with them featuring an animal on the cover, he wanted to name the project after an animal. He came up with Tomcat since he reasoned the animal represented something that could take care of and fend for itself. His wish to see an animal cover eventually came true when O'Reilly published their Tomcat book with a tomcat on the cover.

3.3.3 Android Introduction:

Android is a Linux based operating system it is designed primarily for touch screen mobile devices such as smart phones and tablet computers. The operating system has developed a lot in last 15 years starting from black and white phones to recent smart phones or mini computers. One of the most widely used mobile OS these days is android. The android is software that was founded in Palo Alto of California in 2003.

The android is a powerful operating system and it supports large number of applications in Smartphones. These applications are more comfortable and advanced for the users. The hardware that supports android software is based on ARM architecture platform. The android is an open source operating system means that it's free and any one can use it. The android has got millions of apps available that can help you managing your life one or other way and it is available low cost in market at that reasons android is very popular.

The android development supports with the full java programming language. Even other packages that are API and JSE are not supported. The first version 1.0 of android development kit (SDK) was released in 2008 and latest updated version is jelly bean.

The android is a operating system and is a stack of software components which is divided into five sections and four main layers that is



3.3.3.1 Linux kernel:

The android uses the powerful Linux kernel and it supports wide range of hardware drivers. The kernel is the heart of the operating system that manages input and output requests from software. This provides basic system functionalities like process management, memory management, device management like camera, keypad, display etc the kernel handles all the things. The Linux is really good at networking and it is not necessary to interface it to the peripheral hardware. The kernel itself does not interact directly with the user but rather interacts with the shell and other programs as well as with the hard ware devices on the system.

3.3.3.2 Libraries:

On top of a Linux kernel there is a set of libraries including open source web browser such as webkit, library libc. These libraries are used to play and record audio and video. The SQLite is a data base which is useful for storage and sharing of application data. The SSL libraries are responsible for internet security etc.

3.3.3.3 Android Runtime:

The android runtime provides a key component called Dalvik Virtual Machine which is a kind of java virtual machine. It is specially designed and optimized for android. The Dalvik VM is the process virtual machine in the android operating system. It is software that runs apps on android devices.

The Dalvik VM makes use of Linux core features like memory management and multithreading which is in a java language. The Dalvik VM enables every android application to run its own process. The Dalvik VM executes the files in the .dex format.

3.3.3.4 Application frame work:

The application frame work layer provides many higher level services to applications such as windows manager, view system, package manager, resource manager etc. The application developers are allowed to make use of these services in their application.

3.3.3.5 Applications:

You will find all the android applications at the top layer and you will write your application and install on this layer. Examples of such applications are contacts, books, browsers, services etc. Each application performs a different role in the overall applications.

Advantages:

- Android is Linux based open source operating system , it can be developed by any one
- Easy access to the android apps
- You can replace the battery and mass storage, disk drive and UDB option
- Its supports all Google services
- The operating system is able to inform you of a new SMS and Emails or latest updates.
- It supports Multitasking
- Android phone can also function as a router to share internet
- Its free to customize
- Can install a modified ROM
- Its supports 2D and 3D graphics

CHAPTER 4

4.1 Project Purpose and Scope

4.1 Purpose

The main aim of this project is to achieve the cold start and long tail problems and recommend the products using social data.

4.2 Project Scope

Based on the problems due to cold start and long tail, products in the ecommerce retailers getting sold out so soon . And some kind of products remains stagnant for a long duration. Thus to overcome these problems we are going to track the selling of products as well as recommending the products to the customers of same kind based on the previous purchases by using clustering and classification we achieve this First user register the details like name ,password,address and registers then login with valid credentials then the large amount of transactional data collected from customer for segmentation and classification and these data are preprocessed and cluster and classify the data cold product classified based on sale and long tail is classified based product that getting sold frequently after that cold products that are available in the ecommerce site will be recommended to the user to make retailer some profit for the product and finally long tail problem solved by recommending relevant item and product that are not sold for long period gets attractive offer based on the user request.We use hadoop environment to do manipulation with the products dataset.

4.3 Product Perspective

The number of “hits” has been widely regarded as the lifeblood of many web systems, e.g., e-commerce systems, advertising systems and multimedia consumption systems. However, users would not hit an item if they cannot see it, or they are not interested in the item. Recommender system plays a critical role of discovering interesting items from near-infinite inventory and exhibiting them to potential users. Yet, two issues are crippling the recommender systems. One is “how to handle new users”, and the other is “how to surprise users”. The former is well-known as cold-start recommendation. In this paper, we show that the latter can be investigated as long-tail recommendation. We also exploit the benefits of jointly challenging both cold-start

and long-tail recommendation, and propose a novel approach which can simultaneously handle both of them in a unified objective. For the cold-start problem, we learn from side information, e.g., user attributes, user social relationships, etc. Then, we transfer the learned knowledge to new users. For the long-tail recommendation, we decompose the overall interesting items into two parts: a low-rank part for short-head items and a sparse part for long-tail items. The two parts are independently revealed in the training stage, and transferred into the final recommendation for new users. Furthermore, we effectively formulate the two problems into a unified objective and present an iterative optimization algorithm. A fast extension of the method is proposed to reduce the complexity, and extensive theoretical analysis are provided to proof the bounds of our approach. At last, experiments of social recommendation on various real-world datasets, e.g., images, blogs, videos and musics, verify the superiority of our approach compared with the state-of-the-art work.

4.4 System Features

To the best of our knowledge, this is the first work in the community which challenges both cold-start and long-tail recommendation in a unified objective. An iterative optimization algorithm is further proposed to solve it. Since scalability is a main challenge in recommender systems, we propose a fast extension of the method to reduce the computational complexity. Extensive theoretical analysis, including estimation bound and generalization bound, are presented. Experimental results on four real-world datasets, e.g., images, blogs, videos and musics, verify the superiority of our approach compared with the state-of-the-art work. It is worth noting that our recommendation is beyond accuracy, we also keep an eye on the diversity and serendipity of the recommendations.

4.5 Design and Implementation Constraints

4.5.1 Constraints in Analysis

- ◆ Constraints as Informal Text
- ◆ Constraints as Operational Restrictions

- ◆ Constraints Integrated in Existing Model Concepts
- ◆ Constraints as a Separate Concept
- ◆ Constraints Implied by the Model Structure

4.5.2 Constraints in Design

- ◆ Determination of the Involved Classes
- ◆ Determination of the Involved Objects
- ◆ Determination of the Involved Actions
- ◆ Determination of the Require Clauses
- ◆ Global actions and Constraint Realization

4.5.3 Constraints in Implementation

A hierarchical structuring of relations may result in more classes and a more complicated structure to implement. Therefore it is advisable to transform the hierarchical relation structure to a simpler structure such as a classical flat one. It is rather straightforward to transform the developed hierarchical model into a bipartite, flat model, consisting of classes on the one hand and flat relations on the other. Flat relations are preferred at the design level for reasons of simplicity and implementation ease. There is no identity or functionality associated with a flat relation. A flat relation corresponds with the relation concept of entity-relationship modeling and many object oriented methods.

4.6 Other Nonfunctional Requirements

4.6.1 Performance Requirements

The application at this side controls and communicates with the following three main general components.

- embedded browser in charge of the navigation and accessing to the web service;
- Server Tier: The server side contains the main parts of the functionality of the proposed architecture. The components at this tier are the following.

Web Server, Security Module, Server-Side Capturing Engine, Preprocessing Engine, Database System, Verification Engine, Output Module.

4.6.2 Safety Requirements

1. The software may be safety-critical. If so, there are issues associated with its integrity level
2. The software may not be safety-critical although it forms part of a safety-critical system. For example, software may simply log transactions.
3. If a system must be of a high integrity level and if the software is shown to be of that integrity level, then the hardware must be at least of the same integrity level.
4. There is little point in producing 'perfect' code in some language if hardware and system software (in widest sense) are not reliable.
5. If a computer system is to run software of a high integrity level then that system should not at the same time accommodate software of a lower integrity level.

6. Systems with different requirements for safety levels must be separated.
7. Otherwise, the highest level of integrity required must be applied to all systems in the same environment.

CHAPTER 5

5.1 Architecture Diagram:

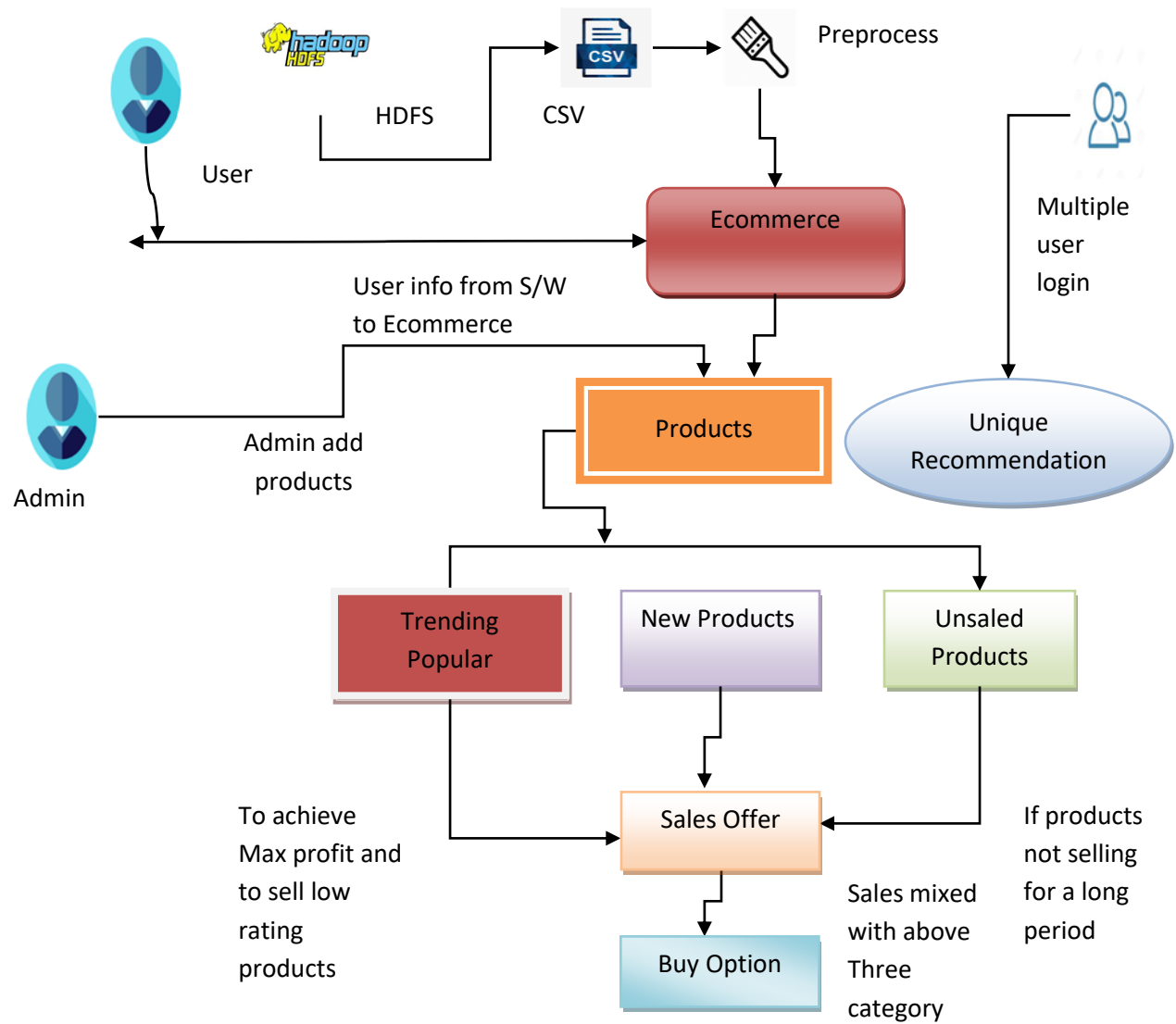


Fig: 5.1

5.2 Sequence Diagram:

A Sequence diagram is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of Message Sequence diagrams are sometimes called event diagrams, event sceneries and timing diagram.

5.3 Use Case Diagram:

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of software engineering. The standard is managed and was created by the Object Management Group. UML includes a set of graphic notation techniques to create visual models of software intensive systems. This language is used to specify, visualize, modify, construct and document the artifacts of an object oriented software intensive system under development.

5.3.1. USECASE DIAGRAM

A Use case Diagram is used to present a graphical overview of the functionality provided by a system in terms of actors, their goals and any dependencies between those use cases.

Use case diagram consists of two parts:

Use case: A use case describes a sequence of actions that provided something of measurable value to an actor and is drawn as a horizontal ellipse.

Actor: An actor is a person, organization or external system that plays a role in one or more interaction with the system.

5.4 Activity Diagram:

Activity diagram is a graphical representation of workflows of stepwise activities and actions with support for choice, iteration and concurrency. An activity diagram shows the overall flow of control.

The most important shape types:

- Rounded rectangles represent activities.
- Diamonds represent decisions.
- Bars represent the start or end of concurrent activities.
- A black circle represents the start of the workflow.
- An encircled circle represents the end of the workflow.

5.5 Collaboration Diagram:

UML Collaboration Diagrams illustrate the relationship and interaction between software objects. They require use cases, system operation contracts and domain model to already exist. The collaboration diagram illustrates messages being sent between classes and objects.

5.6 DATA FLOW DIAGRAM:

A Data Flow Diagram (DFD) is a graphical representation of the “flow” of data through an information system, modeling its aspects. It is a preliminary step used to create an overview of the system which can later be elaborated DFDs can also be used for visualization of data processing.

Level 0:

Level 1:

Level 2:

Level 3:

5.7 Class Diagram

A Class diagram in the Unified Modeling Language is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

CHAPTER 6

SYSTEM DESIGN

Initially user need to create their own account and register their basic detail in the server. Here whereas the Database server will maintain the user personal detail and all the transaction detail which are processed by the user. By the way user need to register into the bank application. A new user account number will be generated where user can deposit initial amount. A huge amount of transactional data has to be collected for customer segmentation and products classification. From the collected dataset the long tail products and cold products will be clustered and classified based on the sales data. Here the most popular products which are getting sold in the online markets is classified which comes under long tail group. Whereas some products which are popular once in trend but was not sold for recent days , these criteria are considered and data preprocess will be done. For a group of customers who came into ecommerce application their previous transactions will be obviously null. For this new customers our recommendation system will provide some valuable products recommendation to attain margin for the retailers. In retailers perspective some cold products will be pushed to recommender platform for the new customers. Thus we can solve the cold start problem by recommending the users some products from cold product category. In-order to solve the long tail problem from the categorized group of products, most selling products has to be collected and some product descriptions will be compared and based on the matching patterns these relevant products will be recommended to the customer. In-order to push the unsold products for a long period based on the customer's request some attractive offers should be promoted. So we implement a product combo offer logic to make the sale possible . Thus long tail and cold start problem has been handled.

6.1 MODULES

- User Authentication
- Data Preprocessing
- Cold Product Recommendation

Combo Products Recommendation

6.2 MODULE EXPLANATION:

6.2.1 User Authentication

Initially user need to create their own account and register their basic detail in the server. Here whereas the Database server will maintain the user personal detail and all the transaction detail which are processed by the user. By the way user need to register into the bank application. A new user account number will be generated where user can deposit initial amount

Data preprocessing

A huge amount of transactional data has to be collected for customer segmentation and products classification. From the collected dataset the long tail products and cold products will be clustered and classified based on the sales data. Here the most popular products which are getting sold in the online markets is classified which comes under long tail group. Whereas some products which are popular once in trend but was not sold for recent days , these criteria are considered and data preprocess will be done.

Cold Product Recommendation

For a group of customers who came into ecommerce application their previous transactions will be obviously null. For this new customers our recommendation system will provide some valuable products recommendation to attain margin for the retailers. In retailers perspective some cold products will be pushed to recommender platform for the new customers. Thus we can solve the cold start problem by recommending the users some products from cold product category.

Combo Products Recommendation

In-order to solve the long tail problem from the categorized group of products, most selling products has to be collected and some product descriptions will be compared and based on the matching

patterns these relevant products will be recommended to the customer. In-order to push the unsold products for a long period based on the customer's request some attractive offers should be promoted. So we implement a product combo offer logic to make the sale possible . Thus long tail and cold start problem has been handled.

CHAPTER 7

CODING AND TESTING

7.1 CODING

Once the design aspect of the system is finalizes the system enters into the coding and testing phase. The coding phase brings the actual system into action by converting the design of the system into the code in a given programming language. Therefore, a good coding style has to be taken whenever changes are required it easily screwed into the system.

7.2 CODING STANDARDS

Coding standards are guidelines to programming that focuses on the physical structure and appearance of the program. They make the code easier to read, understand and maintain. This phase of the system actually implements the blueprint developed during the design phase. The coding specification should be in such a way that any programmer must be able to understand the code and can bring about changes whenever felt necessary. Some of the standard needed to achieve the above-mentioned objectives are as follows:

Program should be simple, clear and easy to understand.

Naming conventions

Value conventions

Script and comment procedure

Message box format

Exception and error handling

7.2.1 NAMING CONVENTIONS

Naming conventions of classes, data member, member functions, procedures etc., should be **self-descriptive**. One should even get the meaning and scope of the variable by its name. The conventions are adopted for **easy understanding** of the intended message by the user. So it is customary to follow the conventions. These conventions are as follows:

Class names

Class names are problem domain equivalence and begin with capital letter and have mixed cases.

Member Function and Data Member name

Member function and data member name begins with a lowercase letter with each subsequent letters of the new words in uppercase and the rest of letters in lowercase.

7.2.2 VALUE CONVENTIONS

Value conventions ensure values for variable at any point of time. This involves the following:

- Proper default values for the variables.
- Proper validation of values in the field.
- Proper documentation of flag values.

7.2.3 SCRIPT WRITING AND COMMENTING STANDARD

Script writing is an art in which indentation is utmost important. Conditional and looping statements are to be properly aligned to facilitate easy understanding. Comments are included to minimize the number of surprises that could occur when going through the code.

7.2.4 MESSAGE BOX FORMAT

When something has to be prompted to the user, he must be able to understand it properly. To achieve this, a specific format has been adopted in displaying messages to the user. They are as follows:

- X – User has performed illegal operation.
- ! – Information to the user.

7.3 TEST PROCEDURE

SYSTEM TESTING

Testing is performed to identify errors. It is used for quality assurance. Testing is an integral part of the entire development and maintenance process. The goal of the testing during phase is to verify that the specification has been accurately and completely incorporated into the design, as well as to ensure the correctness of the design itself. For example the design must not have any logic faults in the design is detected before coding commences, otherwise the cost of fixing the faults will be considerably higher as reflected. Detection of design faults can be achieved by means of inspection as well as walkthrough.

Testing is one of the important steps in the software development phase. Testing checks for the errors, as a whole of the project testing involves the following test cases:

- Static analysis is used to investigate the structural properties of the Source code.
- Dynamic testing is used to investigate the behavior of the source code by executing the program on the test data.

7.4 TEST DATA AND OUTPUT

7.4.1 UNIT TESTING

Unit testing is conducted to verify the functional performance of each modular component of the software. Unit testing focuses on the smallest unit of the software design (i.e.), the module. The white-box testing techniques were heavily employed for unit testing.

7.4.2 FUNCTIONAL TESTS

Functional test cases involved exercising the code with nominal input values for which the expected results are known, as well as boundary values and special values, such as logically related inputs, files of identical elements, and empty files.

Three types of tests in Functional test:

- Performance Test
- Stress Test
- Structure Test

7.4.3 PERFORMANCE TEST

It determines the amount of execution time spent in various parts of the unit, program throughput, and response time and device utilization by the program unit.

7.4.4 STRESS TEST

Stress Test is those test designed to intentionally break the unit. A Great deal can be learned about the strength and limitations of a program by examining the manner in which a programmer in which a program unit breaks.

7.4.5 STRUCTURED TEST

Structure Tests are concerned with exercising the internal logic of a program and traversing particular execution paths. The way in which White-Box test strategy was employed to ensure that the test cases could Guarantee that all independent paths within a module have been have been exercised at least once.

- Exercise all logical decisions on their true or false sides.
- Execute all loops at their boundaries and within their operational bounds.
- Exercise internal data structures to assure their validity.
- Checking attributes for their correctness.
- Handling end of file condition, I/O errors, buffer problems and textual errors in output information

7.4.6 INTEGRATION TESTING

Integration testing is a systematic technique for construction the program structure while at the same time conducting tests to uncover errors associated with interfacing. i.e., integration testing is the complete testing of the set of modules which makes up the product. The objective is to take untested modules and build a program structure tester should identify critical modules. Critical modules should be tested as early as possible. One approach is to wait until all the units have passed testing, and then combine them and then tested. This approach is evolved from unstructured testing of small programs. Another strategy is to construct the product in increments of tested units. A small set of modules are integrated together and tested, to which another module is added and tested in combination. And so on. The advantages of this approach are that, interface dispenses can be easily found and corrected.

The major error that was faced during the project is linking error. When all the modules are combined the link is not set properly with all support files. Then we checked out for interconnection and the links. Errors are localized to the new module and its intercommunications. The product development can be staged, and

modules integrated in as they complete unit testing. Testing is completed when the last module is integrated and tested.

7.5 TESTING TECHNIQUES / TESTING STRATEGIES

7.5.1 TESTING

Testing is a process of executing a program with the intent of finding an error. A good test case is one that has a high probability of finding an as-yet –undiscovered error. A successful test is one that uncovers an as-yet- undiscovered error. System testing is the stage of implementation, which is aimed at ensuring that the system works accurately and efficiently as expected before live operation commences. It verifies that the whole set of programs hang together. System testing requires a test consists of several key activities and steps for run program, string, system and is important in adopting a successful new system. This is the last chance to detect and correct errors before the system is installed for user acceptance testing.

The software testing process commences once the program is created and the documentation and related data structures are designed. Software testing is essential for correcting errors. Otherwise the program or the project is not said to be complete. Software testing is the critical element of software quality assurance and represents the ultimate the review of specification design and coding. Testing is the process of executing the program with the intent of finding the error. A good test case design is one that as a probability of finding an yet undiscovered error. A successful test is one that uncovers an yet undiscovered error. Any engineering product can be tested in one of the two ways:

7.5.1.1 WHITE BOX TESTING

This testing is also called as Glass box testing. In this testing, by knowing the specific functions that a product has been design to perform test can be conducted that demonstrate each function is fully operational at the same time searching for errors in each function. It is a test case design method that uses the control structure of the procedural design to derive test cases. Basis path testing is a white box testing.

Basis path testing:

- Flow graph notation
- Cyclometric complexity
- Deriving test cases
- Graph matrices Control

7.5.1.2 BLACK BOX TESTING

In this testing by knowing the internal operation of a product, test can be conducted to ensure that “all gears mesh”, that is the internal operation performs according to specification and all internal components have been adequately exercised. It fundamentally focuses on the functional requirements of the software.

The steps involved in black box test case design are:

- Graph based testing methods
- Equivalence partitioning
- Boundary value analysis
- Comparison testing

7.5.2 SOFTWARE TESTING STRATEGIES:

A software testing strategy provides a road map for the software developer. Testing is a set activity that can be planned in advance and conducted systematically. For this reason a template for software testing a set of steps into which we can place specific test case design methods should be strategy should have the following characteristics:

- Testing begins at the module level and works “outward” toward the integration of the entire computer based system.

- Different testing techniques are appropriate at different points in time.
- The developer of the software and an independent test group conducts testing.
- Testing and Debugging are different activities but debugging must be accommodated in any testing strategy.

7.5.2.1 INTEGRATION TESTING:

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with. Individual modules, which are highly prone to interface errors, should not be assumed to work instantly when we put them together. The problem of course, is “putting them together”- interfacing. There may be the chances of data lost across on another’s sub functions, when combined may not produce the desired major function; individually acceptable impression may be magnified to unacceptable levels; global data structures can present problems.

7.5.2.2 PROGRAM TESTING:

The logical and syntax errors have been pointed out by program testing. A syntax error is an error in a program statement that in violates one or more rules of the language in which it is written. An improperly defined field dimension or omitted keywords are common syntax error. These errors are shown through error messages generated by the computer. A logic error on the other hand deals with the incorrect data fields, out-off-range items and invalid combinations. Since the compiler s will not deduct logical error, the programmer must examine the output. Condition testing exercises the logical conditions contained in a module. The possible types of elements in a condition include a Boolean operator, Boolean variable, a pair of Boolean parentheses A relational operator or on arithmetic expression. Condition testing method focuses on testing each condition in the program the purpose of condition test is to deduct not only errors in the condition of a program but also other a errors in the program.

7.5.2.3 SECURITY TESTING:

Security testing attempts to verify the protection mechanisms built in to a system well, in fact, protect it from improper penetration. The system security must be tested for invulnerability from frontal attack must also be tested for invulnerability from rear attack. During security, the tester places the role of individual who desires to penetrate system.

7.5.2.4 VALIDATION TESTING

At the culmination of integration testing, software is completely assembled as a package. Interfacing errors have been uncovered and corrected and a final series of software test-validation testing begins. Validation testing can be defined in many ways, but a simple definition is that validation succeeds when the software functions in manner that is reasonably expected by the customer. Software validation is achieved through a series of black box tests that demonstrate conformity with requirement. After validation test has been conducted, one of two conditions exists.

- * The function or performance characteristics confirm to specifications and are accepted.
- * A validation from specification is uncovered and a deficiency created.

Deviation or errors discovered at this step in this project is corrected prior to completion of the project with the help of the user by negotiating to establish a method for resolving deficiencies. Thus the proposed system under consideration has been tested by using validation testing and found to be working satisfactorily. Though there were deficiencies in the system they were not catastrophic

7.5.2.5 USER ACCEPTANCE TESTING

User acceptance of the system is key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with prospective system and user at

the time of developing and making changes whenever required. This is done in regarding to the following points.

- Input screen design.
- Output screen design.

Source Code

```
package com.ecommerce.service;
```

```
import java.util.List;
```

```
import org.hibernate.Session;
```

```
import org.hibernate.SessionFactory;
```

```
import org.hibernate.Transaction;
```

```
import com.ecommerce.model.Product;
```

```
import com.ecommerce.model.Purchased;
```

```
import com.ecommerce.model.User;
```

```
import com.ecommerce.util.EcommerceConstant;
```

```
import com.ecommerce.util.HibernateConfiguration;
```

```
public class AdminImpl implements Admin {
```

```
private static AdminImpl adminImpl = null;
```

```
private AdminImpl() {  
  
}
```

```
SessionFactory factory = HibernateConfiguration.getSessionFactory();
```

```
Session session = null;
```

```
Transaction transaction = null;
```

```
@Override
```

```
public boolean adminLogin(String userName, String pass) {
```

```
    // TODO Auto-generated method stub
```

```
    if (userName != null && pass != null) {
```

```
        if (userName.equals(EcommerceConstant.ADMIN_NAME) &&  
pass.equals(EcommerceConstant.ADMIN_PASS)) {
```

```
            return EcommerceConstant.SUCCESS;
```

```
        }
```

```
    }
```

```
    return EcommerceConstant.FAIL;
```

```
}
```

@Override

```
public void addProduct(Product product) {  
  
    // TODO Auto-generated method stub  
  
    session = factory.openSession();  
  
    transaction = session.beginTransaction();  
  
    session.save(product);  
  
    session.getTransaction().commit();  
  
    session.close();  
  
}
```

@Override

```
public List<Purchased> getAllPurchasedProduct() {  
  
    // TODO Auto-generated method stub  
  
    session = factory.openSession();  
  
    transaction = session.beginTransaction();  
  
    List<Purchased> data = session.createQuery("from Purchased",  
Purchased.class).list();  
  
    session.getTransaction().commit();
```

```
        session.close();

        return data;
    }
}
```

@Override

```
public List<User> getAllUser() {

    // TODO Auto-generated method stub

    session = factory.openSession();

    transaction = session.beginTransaction();

    List<User> data = session.createQuery("from User", User.class).list();

    session.getTransaction().commit();

    session.close();

    return data;
}
```

@Override

```
public List<Product> getAllProduct() {

    // TODO Auto-generated method stub

    session = factory.openSession();

    transaction = session.beginTransaction();
```

```

        List<Product> data = session.createQuery("from Product",
Product.class).list();

        session.getTransaction().commit();

        session.close();

        return data;
    }

    /**
     * To get The Instance of AdminImpl
     *
     * @return {@link AdminImpl}
     */
    public static AdminImpl getInstance() {

        if (adminImpl == null) {

            adminImpl = new AdminImpl();

        }

        return adminImpl;
    }
}

```

```
package com.ecommerce.service;
```

```
import java.util.List;
```

```
import com.ecommerce.model.Product;
```

```
import com.ecommerce.model.Purchased;
```

```
import com.ecommerce.model.User;
```

```
public interface Admin {
```

```
    boolean adminLogin(String userName, String pass);
```

```
    void addProduct(Product product);
```

```
    List<Purchased> getAllPurchasedProduct();
```

```
    List<User> getAllUser();
```

```
    List<Product> getAllProduct();
```

```
}
```

```
package com.ecommerce.service;
```

```
import java.util.List;
```

```
import com.ecommerce.model.Cart;  
  
import com.ecommerce.model.Product;  
  
import com.ecommerce.model.Purchased;  
  
import com.ecommerce.model.User;  
  
import com.ecommerce.model.Userpoi;  
  
  
public interface Customer {  
  
  
  
  
  
  
  
  
  
    boolean createUser(User user);  
  
  
  
  
  
  
  
  
  
    boolean loginUser(String userName, String pass);  
  
  
  
  
  
  
  
  
  
    void addToCart(Cart cart);  
  
  
  
  
  
  
  
  
  
    void removeFromCart(Cart cart);  
  
  
  
  
  
  
  
  
  
    List<Cart> getAllCartProduct(String userName);  
  
  
  
  
  
  
  
  
  
    void addPurchasedProduct(Purchased purchased);
```

List<Purchased> getAllTransaction(String userName);

Product getProductById(int id);

void addUserPoi(Userpoi userpoi);

void addReview(int id, String review);

List<Product> getProductByCat(String product);

List<Product> getColdProduct(String product);

List<Product> getLongTailProduct(String product);

String getUserPoi(String name);

List<Userpoi> getAllUserPOI(String name);

}

```
package com.ecommerce.service;  
  
import java.util.List;  
  
import javax.persistence.criteria.CriteriaBuilder;  
import javax.persistence.criteria.CriteriaQuery;  
import javax.persistence.criteria.Root;  
  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.Transaction;  
import org.hibernate.query.Query;  
  
import com.ecommerce.model.Cart;  
import com.ecommerce.model.Product;  
import com.ecommerce.model.Purchased;  
import com.ecommerce.model.User;  
import com.ecommerce.model.Userpoi;  
import com.ecommerce.util.HibernateConfiguration;
```

```
public class CustomerImpl implements Customer {
```

```
    public static CustomerImpl customerImpl = null;
```

```
    SessionFactory factory = HibernateConfiguration.getSessionFactory();
```

```
    Session session = null;
```

```
    Transaction transaction = null;
```

```
    private CustomerImpl() {
```

```
    }
```

```
@Override
```

```
public boolean createUser(User user) {
```

```
    // TODO Auto-generated method stub
```

```
    session = factory.openSession();
```

```
    transaction = session.beginTransaction();
```

```
    CriteriaBuilder builder = session.getCriteriaBuilder();
```

```
    CriteriaQuery<User> query = builder.createQuery(User.class);
```

```
    Root<User> root = query.from(User.class);
```

```
    query.select(root).where(builder.equal(root.get("email"), user.getEmail()));
```

```

    Query<User> q = session.createQuery(query);

    System.out.println(q.toString());

    List<User> userObj = q.getResultList();

    System.out.println(userObj.toString());

    if (userObj.isEmpty()) {

        session.save(user);

        session.getTransaction().commit();

        return true;

    }

    transaction.commit();

    return false;

}

```

@Override

```

public boolean loginUser(String userName, String pass) {

    // TODO Auto-generated method stub

    session = factory.openSession();

    transaction = session.beginTransaction();

    CriteriaBuilder builder = session.getCriteriaBuilder();

    CriteriaQuery<User> query = builder.createQuery(User.class);

```

```

Root<User> root = query.from(User.class);

query.select(root).where(builder.equal(root.get("email"), userName));

Query<User> q = session.createQuery(query);

System.out.println(q.toString());

List<User> userObj = q.getResultList();

System.out.println(userObj.toString());

if (!userObj.isEmpty()) {

    User user = userObj.get(0);

    if (user.getPass().equals(pass)) {

        return true;

    }

    return false;

}

transaction.commit();

return false;

}

```

@Override

```

public void addToCart(Cart cart) {

    // TODO Auto-generated method stub

```

```

    session = factory.openSession();

    transaction = session.beginTransaction();

    session.save(cart);

    session.getTransaction().commit();

    session.close();

}

```

@Override

```

public void removeFromCart(Cart cart) {

    // TODO Auto-generated method stub

    session = factory.openSession();

    transaction = session.beginTransaction();

    CriteriaBuilder builder = session.getCriteriaBuilder();

    CriteriaQuery<Cart> query = builder.createQuery(Cart.class);

    Root<Cart> root = query.from(Cart.class);

    query.select(root).where(builder.equal(root.get("productId"),
cart.getProductId()),

        builder.equal(root.get("Username"), cart.getUsername()));

    Query<Cart> q = session.createQuery(query);

```

```

        System.out.println(q.toString());

        Cart removecart = q.getSingleResult();

        session.delete(removecart);

        session.getTransaction().commit();

    }

```

@Override

```

public List<Cart> getAllCartProduct(String name) {

    // TODO Auto-generated method stub

    session = factory.openSession();

    transaction = session.beginTransaction();

    CriteriaBuilder builder = session.getCriteriaBuilder();

    CriteriaQuery<Cart> query = builder.createQuery(Cart.class);

    Root<Cart> root = query.from(Cart.class);

    query.select(root).where(builder.equal(root.get("Username"), name));

    Query<Cart> q = session.createQuery(query);

    System.out.println(q.toString());

    List<Cart> cartObj = q.getResultList();

    transaction.commit();

    return cartObj;
}

```

```
}
```

@Override

```
public void addPurchasedProduct(Purchased purchased) {
```

```
    // TODO Auto-generated method stub
```

```
    session = factory.openSession();
```

```
    transaction = session.beginTransaction();
```

```
    session.save(purchased);
```

```
    session.getTransaction().commit();
```

```
    session.close();
```

```
}
```

@Override

```
public List<Purchased> getAllTransaction(String userName) {
```

```
    // TODO Auto-generated method stub
```

```
    session = factory.openSession();
```

```
    transaction = session.beginTransaction();
```

```
    CriteriaBuilder builder = session.getCriteriaBuilder();
```

```
    CriteriaQuery<Purchased> query = builder.createQuery(Purchased.class);
```

```
    Root<Purchased> root = query.from(Purchased.class);
```

```

        query.select(root).where(builder.equal(root.get("UserName"), userName));

        Query<Purchased> q = session.createQuery(query);

        System.out.println(q.toString());

        List<Purchased> purchasedObj = q.getResultList();

        transaction.commit();

        return purchasedObj;
    }

```

@Override

```

public Product getProductById(int id) {

    // TODO Auto-generated method stub

    session = factory.openSession();

    transaction = session.beginTransaction();

    Product product = session.get(Product.class, id);

    transaction.commit();

    return product;
}

```

@Override

```

public void addUserPoi(Userpoi userpoi) {

```

```

// TODO Auto-generated method stub

session = factory.openSession();

transaction = session.beginTransaction();

CriteriaBuilder builder = session.getCriteriaBuilder();

CriteriaQuery<Userpoi> query = builder.createQuery(Userpoi.class);

Root<Userpoi> root = query.from(Userpoi.class);

query.select(root).where(builder.equal(root.get("User"), userpoi.getUser()),

        builder.equal(root.get("Product"), userpoi.getProduct()));

Query<Userpoi> q = session.createQuery(query);

System.out.println(q.toString());

List<Userpoi> userObj = q.getResultList();

if (!userObj.isEmpty()) {

    Userpoi uPoi = (Userpoi) userObj.get(0);

    uPoi.setHitcount(

        String.valueOf(Integer.parseInt(uPoi.getHitcount()) +

Integer.parseInt(userpoi.getHitcount())));

    session.merge(uPoi);

} else {

    session.save(userpoi);

}

```

```

        transaction.commit();
    }

    @Override
    public List<Product> getProductByCat(String product) {

        // TODO Auto-generated method stub

        session = factory.openSession();

        transaction = session.beginTransaction();

        CriteriaBuilder builder = session.getCriteriaBuilder();

        CriteriaQuery<Product> query = builder.createQuery(Product.class);

        Root<Product> root = query.from(Product.class);

        query.select(root).where(builder.equal(root.get("productName"), product));

        Query<Product> q = session.createQuery(query);

        System.out.println(q.toString());

        List<Product> productObj = q.getResultList();

        transaction.commit();

        return productObj;
    }

```

@Override

```
public List<Product> getColdProduct(String product) {  
  
    // TODO Auto-generated method stub  
  
    session = factory.openSession();  
  
    transaction = session.beginTransaction();  
  
    CriteriaBuilder builder = session.getCriteriaBuilder();  
  
    CriteriaQuery<Product> query = builder.createQuery(Product.class);  
  
    Root<Product> root = query.from(Product.class);  
  
    query.select(root).where(builder.equal(root.get("productName"), product));  
  
    query.orderBy(builder.asc(root.get("buyers")));  
  
    Query<Product> q = session.createQuery(query);  
  
    q.setMaxResults(5);  
  
    System.out.println(q.toString());  
  
    List<Product> productObj = q.getResultList();  
  
    transaction.commit();  
  
    return productObj;  
}
```

@Override

```
public List<Product> getLongTailProduct(String product) {
```

```

// TODO Auto-generated method stub

session = factory.openSession();

transaction = session.beginTransaction();

CriteriaBuilder builder = session.getCriteriaBuilder();

CriteriaQuery<Product> query = builder.createQuery(Product.class);

Root<Product> root = query.from(Product.class);

query.select(root).where(builder.equal(root.get("productName"), product));

query.orderBy(builder.desc(root.get("buyers")));

Query<Product> q = session.createQuery(query);

q.setMaxResults(5);

System.out.println(q.toString());

List<Product> productObj = q.getResultList();

transaction.commit();

return productObj;
}

```

@Override

```

public String getUserPoi(String name) {

    String productName = null;

    session = factory.openSession();

```

```

transaction = session.beginTransaction();

CriteriaBuilder builder = session.getCriteriaBuilder();

CriteriaQuery<Userpoi> query = builder.createQuery(Userpoi.class);

Root<Userpoi> root = query.from(Userpoi.class);

query.select(root).where(builder.equal(root.get("User"), name));

query.orderBy(builder.desc(root.get("Hitcount")));

Query<Userpoi> q = session.createQuery(query);

q.setMaxResults(1);

System.out.println(q.toString());

List<Userpoi> userObj = q.getResultList();

if (!userObj.isEmpty()) {

    Userpoi uPoi = (Userpoi) userObj.get(0);

    System.out.println("-----" + uPoi.getProduct());

    productName = uPoi.getProduct();

} else {

    productName = "";

}

return productName;

}

```

@Override

```
public List<Userpoi> getAllUserPOI(String name) {  
  
    // TODO Auto-generated method stub  
  
    session = factory.openSession();  
  
    transaction = session.beginTransaction();  
  
    CriteriaBuilder builder = session.getCriteriaBuilder();  
  
    CriteriaQuery<Userpoi> query = builder.createQuery(Userpoi.class);  
  
    Root<Userpoi> root = query.from(Userpoi.class);  
  
    query.select(root).where(builder.equal(root.get("User"), name));  
  
    Query<Userpoi> q = session.createQuery(query);  
  
    System.out.println(q.toString());  
  
    List<Userpoi> purchasedObj = q.getResultList();  
  
    transaction.commit();  
  
    return purchasedObj;  
  
}
```

/**

*** To get The Instance of CustomerImpl**

```
* @return {@link CustomerImpl}
```

```
*/
```

```
public static CustomerImpl getInstance() {
```

```
    if (customerImpl == null) {
```

```
        customerImpl = new CustomerImpl();
```

```
    }
```

```
    return customerImpl;
```

```
}
```

```
@Override
```

```
public void addReview(int id, String review) {
```

```
    session = factory.openSession();
```

```
    transaction = session.beginTransaction();
```

```
    CriteriaBuilder builder = session.getCriteriaBuilder();
```

```
    CriteriaQuery<Purchased> query = builder.createQuery(Purchased.class);
```

```
    Root<Purchased> root = query.from(Purchased.class);
```

```
    query.select(root).where(builder.equal(root.get("Id"), id));
```

```
    Query<Purchased> q = session.createQuery(query);
```

```
    System.out.println(q.toString());
```

```
List<Purchased> userObj = q.getResultList();

if (!userObj.isEmpty()) {

    Purchased uPoi = (Purchased) userObj.get(0);

    uPoi.setReview(review);

    session.merge(uPoi);

} else {

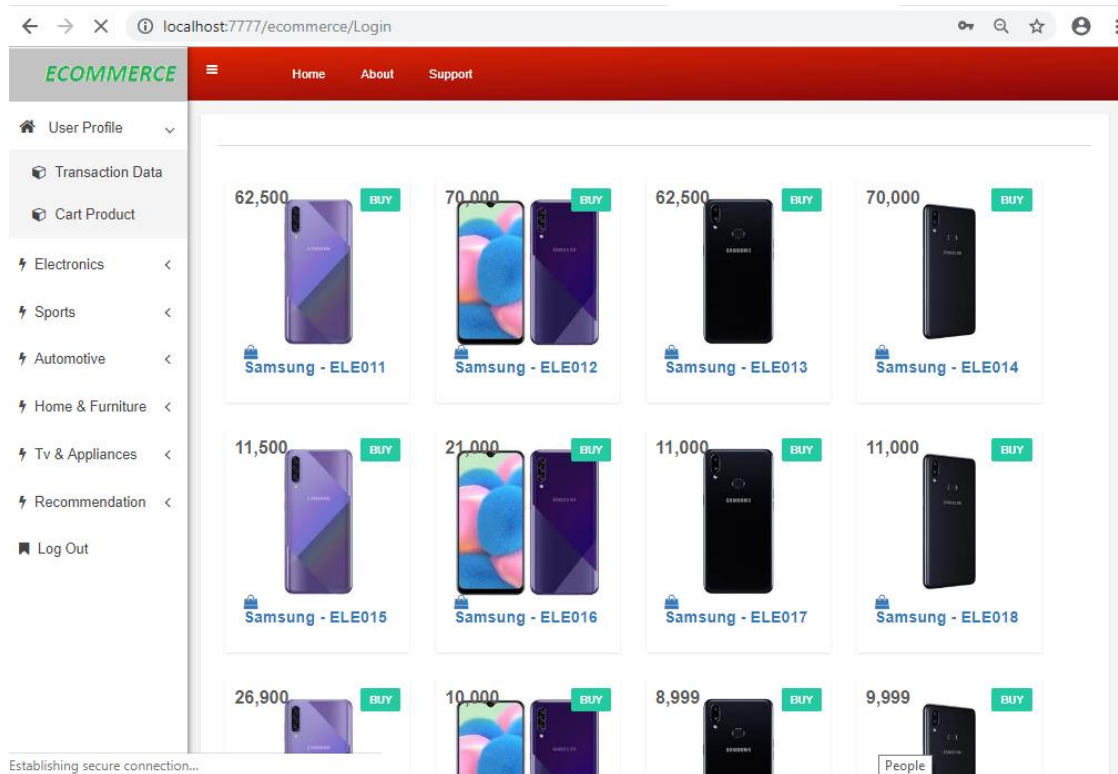
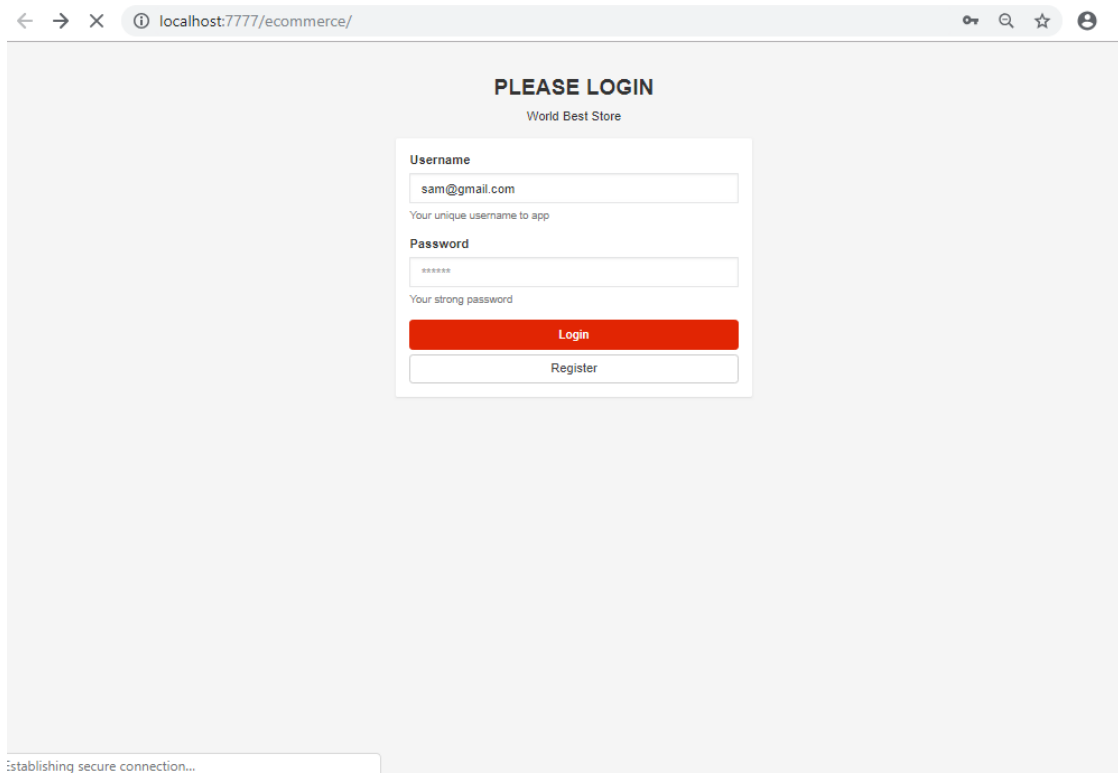
}

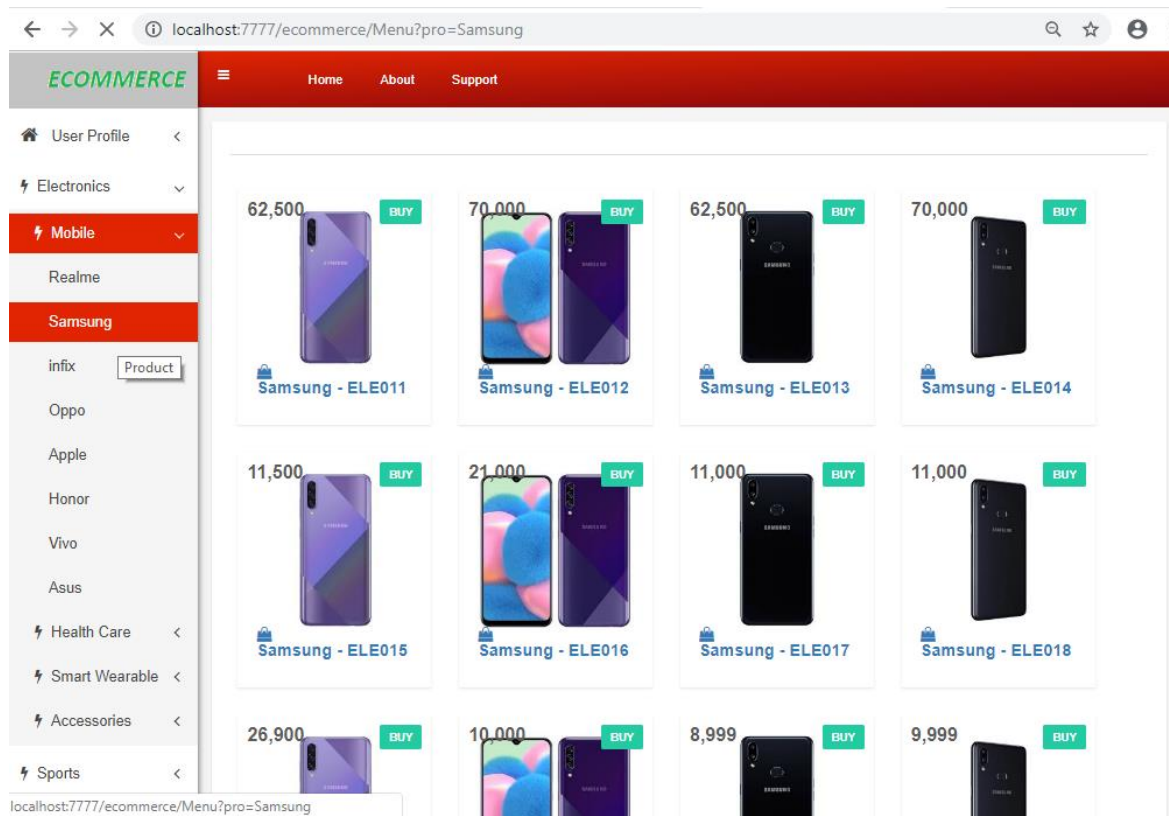
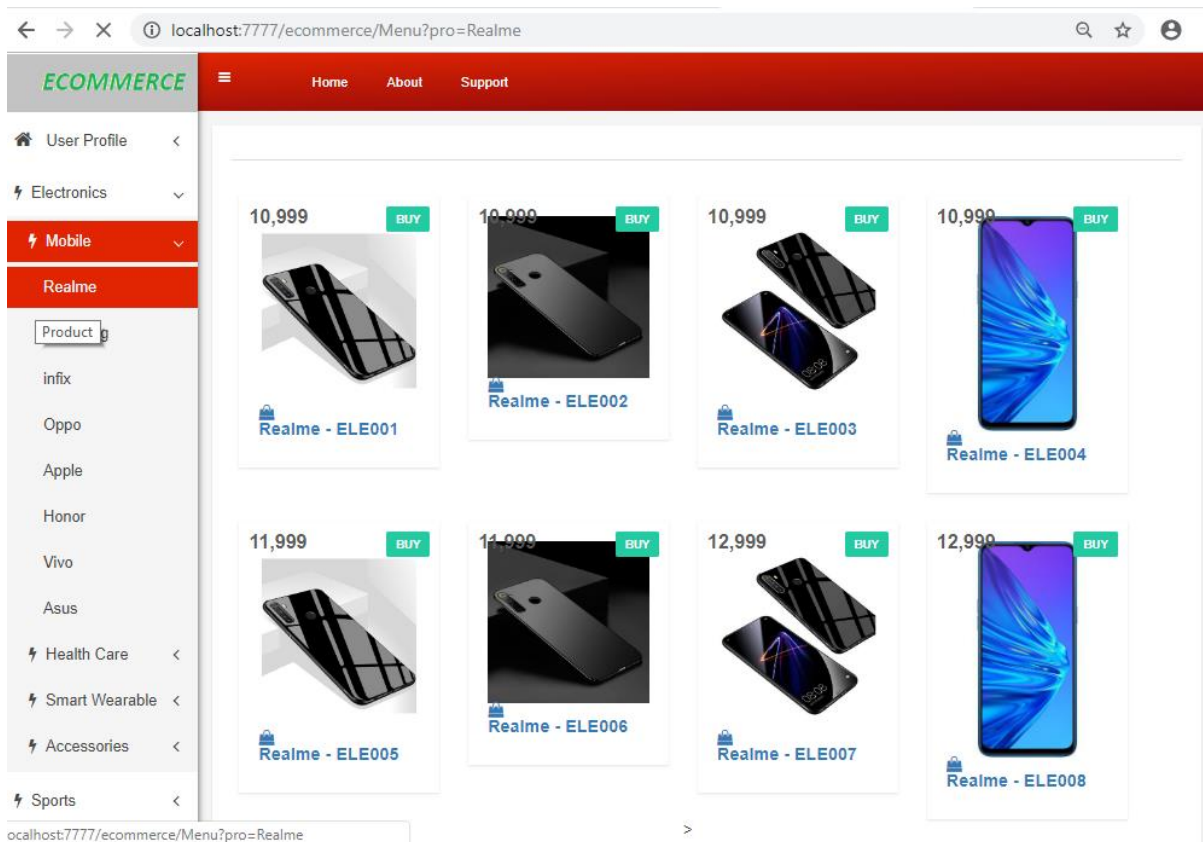
transaction.commit();

}

}
```

Screenshots:





Buy product

← → ↻ ⓘ localhost:7777/ecommerce/Menu?pro=Samsung

ECOMMERCE

Home About Support

User Profile

Transaction Data

Cart Product

Electronics

Sports

Automotive

Home & Furniture

Tv & Appliances

Recommendation

Log Out

62,500 BUY Samsung - ELE011

70,000 BUY Samsung - ELE012

62,500 BUY Samsung - ELE013

70,000 BUY Samsung - ELE014

11,500 BUY Samsung - ELE015

21,000 BUY Samsung - ELE016

11,000 BUY Samsung - ELE017

11,000 BUY Samsung - ELE018

26,900 BUY

10,000 BUY

8,999 BUY

9,999 BUY

localhost:7777/ecommerce/Buy?code=S&amt=11,000&id=20

← → ↻ ⓘ Not secure | 10.0.0.22:7777/BankAdmin/setuserpayment.jsp

Global Bank



HOME CONTACT

Payment

Account No:

Atm Pin:

OPass:

Submit

©GLOBAL TECHNO SOLUTION..

← → ↻ ⚠ Not secure | 10.0.0.22:7777/BankAdmin/setuserpayment.jsp 🔑 ☆ 👤

Global Bank



HOME CONTACT

Payment

Account No: 85380318

Atm Pin: ****

OPass:

Submit

©GLOBAL TECHNO SOLUTION..

← → ↻ ⓘ localhost:7777/ecommerce/LoadCard# 🔑 🔍 ☆ 👤

ECOMMERCE

Home About Support

User Profile

Transaction Data

Cart Product

Electronics

Sports

Automotive

Home & Furniture

Tv & Appliances

Recommendation

Log Out

Product ID	Category	Sub Category	ProductName	Date	Amount	
ELE011	Electronics	Mobile	Samsung	2019-10-16 19:03:04.0	62,500	Buy
ELE011	Electronics	Mobile	Samsung	2019-10-16 19:03:04.0	62,500	Buy

← → × ⓘ localhost:7777/ecommerce/LoadCard 🔍 ☆ 👤 ⋮

ECOMMERCE

☰ Home About Support

🏠 User Profile <

⚡ Electronics <

⚡ Sports <

⚡ Automotive <

⚡ Home & Furniture <

⚡ Tv & Appliances <

⚡ Recommendation ▾

Combo

General Offer

🚪 Log Out

Product ID	Category	Sub Category	ProductName	Date	Amount
>					
Showing 1 to 1 of 1 rows					

Establishing secure connection...

← → ↻ ⓘ localhost:7777/ecommerce/LoadCard 🔍 ☆ 👤 ⋮

ECOMMERCE

☰ Home About Support

🏠 User Profile <

⚡ Electronics <

⚡ Sports <

⚡ Automotive <

⚡ Home & Furniture <

⚡ Tv & Appliances <

⚡ Recommendation ▾

Combo

General Offer Products

🚪 Log Out

Product ID	Category	Sub Category	ProductName	Date	Amount
>					
Showing 1 to 1 of 1 rows					

localhost:7777/ecommerce/comboGene.jsp

localhost:7777/ecommerce/comboGene.jsp

Combo Generation

Select Your Preferred Products With Different Category

First Product Realme

Second Product Samsung

Generate

localhost:7777/ecommerce/ComboProduct?first=Realme&second=Samsung

ECOMMERCE

Home About Support

- User Profile
- Transaction Data
- Cart Product
- Electronics
- Sports
- Automotive
- Home & Furniture
- Tv & Appliances
- Recommendation
- Log Out

New Added Product

10,999 Realme - ELE001 OFFER	+	62,500 Samsung - ELE011 OFFER	ELE011 + ELE001 10,999 + 62,500 Mrp Price 73499 15 % off Rs 62475 BUY
15,999 Realme - ELE009 OFFER	+	62,500 Samsung - ELE011 OFFER	ELE011 + ELE009 15,999 + 62,500 Mrp Price 78499 15 % off Rs 66725 BUY
10,999 Realme - ELE003 OFFER	+	62,500 Samsung - ELE011 OFFER	ELE011 + ELE003 10,999 + 62,500 Mrp Price 73499 BUY

Service Runner

```

Commons Daemon Service Runner (2)
Combo Product's areELE070AndELE010
Applied Offers are 15% off
-----
Generated Combo Product Are --->{0={com.ecommerce.model.Product@1f53617=com.ecommerce.model.Product@25553d, com.ecommerce.model.Product@12d3599=com.ecommerce.model.Product@25553d, com.ecommerce.model.Product@1078f62=com.ecommerce.model.Product@25553d, com.ecommerce.model.Product@43143=com.ecommerce.model.Product@25553d, com.ecommerce.model.Product@b93321=com.ecommerce.model.Product@25553d, com.ecommerce.model.Product@12e7076=com.ecommerce.model.Product@25553d, com.ecommerce.model.Product@1c30818=com.ecommerce.model.Product@25553d, com.ecommerce.model.Product@18ee0e5=com.ecommerce.model.Product@25553d, com.ecommerce.model.Product@562ea8=com.ecommerce.model.Product@25553d, com.ecommerce.model.Product@494f86=com.ecommerce.model.Product@25553d}, 1={com.ecommerce.model.Product@1f53617=com.ecommerce.model.Product@7e388b, com.ecommerce.model.Product@12d3599=com.ecommerce.model.Product@7e388b, com.ecommerce.model.Product@1078f62=com.ecommerce.model.Product@7e388b, com.ecommerce.model.Product@43143=com.ecommerce.model.Product@7e388b, com.ecommerce.model.Product@b93321=com.ecommerce.model.Product@7e388b, com.ecommerce.model.Product@1c30818=com.ecommerce.model.Product@7e388b, com.ecommerce.model.Product@18ee0e5=com.ecommerce.model.Product@7e388b, com.ecommerce.model.Product@562ea8=com.ecommerce.model.Product@7e388b, com.ecommerce.model.Product@494f86=com.ecommerce.model.Product@7e388b}, 2={com.ecommerce.model.Product@1f53617=com.ecommerce.model.Product@1b1e6a4, com.ecommerce.model.Product@12d3599=com.ecommerce.model.Product@1b1e6a4, com.ecommerce.model.Product@1078f62=com.ecommerce.model.Product@1b1e6a4, com.ecommerce.model.Product@43143=com.ecommerce.model.Product@1b1e6a4, com.ecommerce.model.Product@b93321=com.ecommerce.model.Product@1b1e6a4, com.ecommerce.model.Product@12e7076=com.ecommerce.model.Product@1b1e6a4, com.ecommerce.model.Product@1c30818=com.ecommerce.model.Product@1b1e6a4, com.ecommerce.model.Product@18ee0e5=com.ecommerce.model.Product@1b1e6a4, com.ecommerce.model.Product@562ea8=com.ecommerce.model.Product@1b1e6a4, com.ecommerce.model.Product@494f86=com.ecommerce.model.Product@1b1e6a4}, 3={com.ecommerce.model.Product@1f53617=com.ecommerce.model.Product@605fc, com.ecommerce.model.Product@12d3599=com.ecommerce.model.Product@605fc, com.ecommerce.model.Product@1078f62=com.ecommerce.model.Product@605fc, com.ecommerce.model.Product@43143=com.ecommerce.model.Product@605fc, com.ecommerce.model.Product@b93321=com.ecommerce.model.Product@605fc, com.ecommerce.model.Product@12e7076=com.ecommerce.model.Product@605fc, com.ecommerce.model.Product@1c30818=com.ecommerce.model.Product@605fc, com.ecommerce.model.Product@18ee0e5=com.ecommerce.model.Product@605fc, com.ecommerce.model.Product@562ea8=com.ecommerce.model.Product@605fc, com.ecommerce.model.Product@494f86=com.ecommerce.model.Product@605fc}, 4={com.ecommerce.model.Product@1f53617=com.ecommerce.model.Product@fc23fc, com.ecommerce.model.Product@12d3599=com.ecommerce.model.Product@fc23fc, com.ecommerce.model.Product@1078f62=com.ecommerce.model.Product@fc23fc

```

ECOMMERCE

Home

About

Support

User Profile

Transaction Data

Cart Product

Electronics

Sports

Automotive

Home & Furniture

Tv & Appliances

Recommendation

Log Out

Product ID

Category

Sub Category

ProductName

USer Review

Review

ELE017

Electronics

Mobile

Samsung

Super

Add Review

ELE061 ELE003

Electronics Electronics

Mobile Mobile

Honor Realme

Good

Add Review

Showing 1 to 3 of 3 rows

Establishing secure connection...

MySQL-Front - localhost - bankadmindb (Object Browser)

File Edit Search View Favorites Database Extras Settings Help

Object Browser Data Browser SQL Editor

localhost

- bankadmindb
 - tokengeneration
 - useraccountinfo
 - userinfo
 - crime
 - ecommerce
 - ecs
 - ecsauthority
 - information_schema
 - mysql
 - quantcloud
 - social_book
 - staffcredit
 - test
 - university
 - Hosts
 - Processes
 - Status
 - User
 - Variables

Name	Type	Records	Size	Last Update	Extras	Comment
tokengenera...	InnoDB	~31	16 KB	???		
useraccounti...	InnoDB	~14	16 KB	???		
userinfo	InnoDB	~14	16 KB	???		

MySQL-Front - localhost - bankadmindb.tokengeneration (Data Browser)

File Edit Search View Favorites Database Extras Settings Help

Object Browser Data Browser SQL Editor

0 50 Filter Search

accountno	accholdename	time	productid	productamt	tokenid	status
14543675	1	2015-12-14 18:34:04	B00010TE16	19596	Tk193	failure
14543675	webserver1	2015-12-14 18:54:38	B00062FFHK	99840	Tk242	success
14543675	webserver1	2015-12-15 18:12:47	B000067NNS	23	Tk690	success
14543675	webserver1	2015-12-15 18:17:33	B000067NNS	23	Tk723	success
14543675	webserver0	2015-12-15 19:02:16	B000234I9Y	73	Tk157	success
14543675	webserver0	2015-12-15 19:07:35	B000234I9Y	73	Tk198	success
14543675	webserver0	2015-12-15 19:10:28	B00062FFK2	28896	Tk311	success
14543675	webserver0	2015-12-15 19:12:35	B000234I9Y	73	Tk214	success
14543675	webserver0	2015-12-15 19:16:33	B000234I9Y	73	Tk460	success
14543675	webserver0	2015-12-15 19:20:28	B0002IEDEO	3419	Tk914	success
14543675	webserver0	2015-12-16 11:29:13	B00005NCFH	44	Tk118	success
14543675	webserver0	2015-12-16 12:00:53	B000EFH1NA	52076	Tk186	success
14543675	webserver0	2015-12-16 12:46:25	B000F7UM2Y	35214	Tk923	success
14543675	webserver1	2015-12-16 12:51:26	B000A3RV32	36	Tk608	success

MySQL-Front - localhost - bankadmindb.useraccountinfo (Data Browser)

File Edit Search View Favorites Database Extras Settings Help

Object Browser Data Browser SQL Editor

0 50 Filter Search

accountno	pinpass	amount	opass	transferamount
06797274	3399	86753	#1XA+42	null
14543675	0438	384334	SC1oiK1z	929
30998432	2906	981258	IDvsEEHc	null
72483896	1411	10000	null	null
75372272	7915	527798	null	null
81450999	2035	9689672	null	null
96707515	5455	571544	BmjfwJm7	null
85339945	9192	19775483	LRCMa8/7	null

MySQL-Front - localhost - bankadmindb.userinfo (Data Browser)

File Edit Search View Favorites Database Extras Settings Help

Object Browser Data Browser SQL Editor

localhost

- bankadmindb
 - tokengeneration
 - useraccountinfo
 - userinfo
 - crime
 - ecommerce
 - ecs
 - ecsauthority
 - information_schema
 - mysql

accountno	accountholdema...	mobilenumber	mailid	Address	city	pincode	state
06797274	uma	2345678945	uma@gmail.com	main Road	Chennai G.P.O.	600001	TAMIL NADU
14543675	stephen	9003394322	stef@gmail.com	chennai 1st street	Chennai G.P.O.	600001	TAMIL NADU
30998432	ssss	3456234568	ssss@gmail.com	chennai 1st	Chennai G.P.O.	600001	TAMIL NADU
72483896	webserver2	9750882465	dfd@gmail.com	chennai	Chennai G.P.O.	600001	TAMIL NADU
75372272	webserver1	9003394322	df1@gmail.com	chennai	Chennampatti	638504	TAMIL NADU
81450999	webserver0	9878654311	sss@gmail.com	chennai	Chennai Race Cours	600032	TAMIL NADU
96707515	mmm	9837426352	mmm@gmail.com	chennai	Nungambakkam Hig	600034	TAMIL NADU

MySQL-Front - localhost - ecommerce (Object Browser)

File Edit Search View Favorites Database Extras Settings Help

Object Browser Data Browser SQL Editor

localhost

- bankadmindb
- crime
- ecommerce
 - product_cart
 - product_detail
 - product_trans
 - user
 - user_poi
- ecs
- ecsauthority
- information_schema
- mysql

Name	Type	Records	Size	Last Update	Extras	Comment
product_cart	InnoDB	~8	16 KB	???		
product_detail	InnoDB	~517	112 KB	???		
product_trans	InnoDB	~23	16 KB	???		
user	InnoDB	~9	16 KB	???		
user_poi	MyISAM	46	3,709 B	2/29/2020 10:...		

MySQL-Front - localhost - ecommerce.product_cart (Data Browser)

File Edit Search View Favorites Database Extras Settings Help

Object Browser Data Browser SQL Editor

localhost

- bankadmindb
- crime
- ecommerce
 - product_cart
 - product_detail
 - product_trans
 - user
 - user_poi
- ecs
- ecsauthority

Id	Product_Id	Username	amount
1	3	ediwakar70@gmail.c	10,999
6	45	msd@gmail.com	70,000
7	26	msd@gmail.com	8,999
8	13	a@gmail.com	62,500
10	73	a@gmail.com	12,999
11	3	han@gmail.com	10,999
12	13	sam@gmail.com	62,500

MySQL-Front - localhost - ecommerce.product_detail (Data Browser)

File Edit Search View Favorites Database Extras Settings Help

Object Browser Data Browser SQL Editor

localhost

- bankadmindb
- crime
- ecommerce
 - product_cart
 - product_detail
 - product_trans
 - user
 - user_poi
- ecs
- ecsauthority
- information_schema
- mysql
- quantcloud
- social_book
- staffcredit
- test
- university

Id	ProductId	Category	Subcategory	ProductName	Description	Spec	Rating	Buyers
3	ELE001	Electronics	Mobile	Realme	The redmi 8 comes v	Redmi 8(Ruby Red, €	1.5	0
4	ELE002	Electronics	Mobile	Realme	The redmi 8 comes v	Redmi 8(Sapphire Bl	3	37339
5	ELE003	Electronics	Mobile	Realme	The redmi 8 comes v	Redmi 8(Onyx Black	1.5	24095
6	ELE004	Electronics	Mobile	Realme	The redmi 8 comes v	Redmi 8(Emerald Gre	2.5	45685
7	ELE005	Electronics	Mobile	Realme	Featuring a quad-car	Realme 5s(Crystal P	1	38298
8	ELE006	Electronics	Mobile	Realme	Featuring a quad-car	Realme 5s(Crystal Bl	4.5	35706
9	ELE007	Electronics	Mobile	Realme	Featuring a quad-car	Realme 5s(Crystal P	2	13151
10	ELE008	Electronics	Mobile	Realme	Featuring a quad-car	Realme 5s(Crystal Bl	1	49131
11	ELE009	Electronics	Mobile	Realme	It's time to go big wi	Redmi Note 7 Pro(As	2.5	29199
12	ELE010	Electronics	Mobile	Realme	It's time to go big wi	Redmi Note 7 Pro(N	5	16182
13	ELE011	Electronics	Mobile	Samsung	A truckload of drool	Samsung Galaxy s9	1.5	39857
14	ELE012	Electronics	Mobile	Samsung	A blend of innovative	Samsung Galaxy s9	2	7311
15	ELE013	Electronics	Mobile	Samsung	A truckload of drool	Samsung Galaxy s9	2.5	28248

MySQL-Front - localhost - ecommerce.product_trans (Data Browser)

File Edit Search View Favorites Database Extras Settings Help

Object Browser Data Browser SQL Editor

localhost

- bankadmindb
- crime
- ecommerce
 - product_cart
 - product_detail
 - product_trans
 - user
 - user_poi
- ecs
- ecsauthority

Id	Product_ID	Amount	Date	Username	category	productName	Review	subCategory
1	ELE072	15,599	2020-02-19 22:37:35	ediwakar70@gmail.c	Electronics	Asus	Good	Mobile
2	ELE072	15,599	2020-02-19 23:01:40	ediwakar70@gmail.c	Electronics	Asus	Good	Mobile
3	ELE072	15,599	2020-02-19 23:01:41	ediwakar70@gmail.c	Electronics	Asus	Good	Mobile
4	ELE072	15,599	2020-02-19 23:01:41	ediwakar70@gmail.c	Electronics	Asus	Good	Mobile
5	ELE072	15,599	2020-02-19 23:01:42	ediwakar70@gmail.c	Electronics	Asus	Good	Mobile
6	ELE072	15,599	2020-02-19 23:01:42	ediwakar70@gmail.c	Electronics	Asus	Good	Mobile
7	ELE072	15,599	2020-02-19 23:01:44	ediwakar70@gmail.c	Electronics	Asus	Good	Mobile

MySQL-Front - localhost - ecommerce.user (Data Browser)

File Edit Search View Favorites Database Extras Settings Help

Object Browser Data Browser SQL Editor

localhost

- bankadmindb
- crime
- ecommerce
 - product_cart
 - product_detail
 - product_trans
 - user
 - user_poi
- ecs
- ecsauthority
- information schema

Id	Name	Password	Email	Address
1	Diwakar	diwa123	ediwakar70@gmail.c	Chennai
2	Test	111	test@gmail.com	Chennai
3	msd	msd123	msd@gmail.com	chn
4	a	1234	a@gmail.com	aa
5	hari	1234	hari@gmail.com	ellisroad
6	sam	1234	sam@gmail.com	sam
7	jam	1234	jam@gmail.com	ellisroad
8	oam	1234	oam@gmail.com	ellis

MySQL-Front - localhost - ecommerce.user_poi (Data Browser)

File Edit Search View Favorites Database Extras Settings Help

Object Browser Data Browser SQL Editor

localhost

- bankadmindb
- crime
- ecommerce
 - product_cart
 - product_detail
 - product_trans
 - user
 - user_poi
- ecs
- ecsauthority

id	Hitcount	Product	User
1	2	Asus	ediwakar70@gmail.c
3	1	Apple	ediwakar70@gmail.c
12	2	Car Electronics and	ediwakar70@gmail.c
13	1	Honor	ediwakar70@gmail.c
14	5	Realme	ediwakar70@gmail.c
15	1	Samsung	ediwakar70@gmail.c
16	2	Apple	msd@gmail.com

REFERENCES

1. E. Airodi, D. Blei, S. Fienberg, and E. P. Xing. Mixed membership stochastic blockmodels. *J. Mach. Learn. Res.*, 9:1981–2014, 2008.
2. U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski, and D. Wagner. Maximizing modularity is hard. *Arxivpreprint physics/0608255*, 2006.
3. D. Chakrabarti and C. Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Comput. Surv.*, 38(1):2, 2006.
4. S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *Proceedings of the ACM SIGMOD international conference on Management of data*, 1998.
5. G. Chen, F. Wang, and C. Zhang. Semi-supervised multi-label learning by solving a sylvester equation. In *SDM*, 2008.
6. L. Danon, J. Duch, A. Arenas, and A. D'íaz-guilera. Comparing community structure identification. *J. Stat. Mech*, 2005.
7. R.-E. Fan and C.-J. Lin. A study on threshold selection for multi-label classification. 2007.
8. B. Gallagher, H. Tong, T. Eliassi-Rad, and C. Faloutsos. Using ghost edges for classification in sparsely labeled networks. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2008.
9. S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. pages 452–472, 1990.
10. L. Getoor and B. Taskar, editors. *Introduction to Statistical Relational Learning*. The MIT Press, 2007.
11. M. S. Handcock, A. E. Raftery, and J. M. Tantrum. Model-based clustering for social networks. *Journal Of The Royal Statistical Society Series A*, 2007.
12. P. D. Hoff and M. S. H. Adrian E. Raftery. Latent space approaches to social network analysis. *Journal of the American Statistical Association*, 97(460):1090–1098, 2002

