CS 634

DATA MINING

REPORT

A picture containing toy

Description automatically generated

ROHIT THOLE

[rlt23@njit.edu](mailto:rlt23@njit.edu)

Table of Contents:

1. Introduction
2. Description
   1. Association Rules
   2. Support
   3. Confidence
   4. Working
   5. Frequent Itemset
3. Purpose
4. Requirements
   1. Software Requirements
   2. Software Links
   3. Hardware Requirements
5. System Design
   1. Overview
   2. Database Design
6. Database
   1. Itemlist
   2. Transactions
7. Introduction

The Apriori algorithm was proposed by Agrawal and Srikant in 1994. With the quick growth in e-commerce applications, there is an accumulation of vast quantity of data in months. Data Mining, also known as Knowledge Discovery in Databases(KDD), to find anomalies, correlations, patterns, and trends to predict outcomes.

Apriori algorithm is a classical algorithm in data mining. It is used for mining frequent itemsets and relevant association rules. It is devised to operate on a database containing a lot of transactions, for instance, items brought by customers in a store. It helps customers in buying items with more ease which increases the sales of the markets.

 It has also been used in the field of healthcare for the detection of adverse drug reactions. It produces association rules that indicates what all combinations of medications and patient characteristics lead to ADRs.

**2.0 Description**

**2.1 Association Rules:**

Association rule learning is a prominent and a well-explored method for determining relations among variables in large databases. This is often used for boosting sales in e-commerce.

**2.2 Support:**

The support of an itemset X, supp(X) is the proportion of transaction in the database in which the item X appears. It signifies the popularity of an itemset.

Graphical user interface, text, application, Word

Description automatically generated

If the sales of a particular product (item) above a certain proportion have a meaningful effect on profits, that proportion can be considered as the support threshold. Furthermore, we can identify itemsets that have support values beyond this threshold as significant itemsets.

**2.3 Confidence:**

It signifies the likelihood of item Y being purchased when item X is purchased.

**A picture containing text

Description automatically generated**

It can also be interpreted as the conditional probability P(Y|X), i.e. the probability of finding the itemset Y in transactions given the transaction already contains X.

**2.4 Working:**

A key concept in Apriori algorithm is the anti-monotonicity of the support measure. It assumes that

1. All subsets of a frequent itemset must be frequent
2. Similarly, for any infrequent itemset, all its supersets must be infrequent too

**2.5 Frequent Itemset:**

A frequent itemset is an itemset whose support is greater than some user-specified minimum support (denoted Lk, where k is the size of the itemset).

**3.Purpose:**

The purpose of this project is to implement a system with The Apriori Algorithm, that can generate all the association rules along with Support and Confidence for any given dataset. Then, implement the same using Brute Force to compare the efficiency between Apriori Algorithm and Brute Force. This project gives a deeper insight into customer’s buying patterns which will help businesses to increase their sales.

**4.Requirements**

**4.1 Software Requirements**:

* Java Development Kit – Version 1.8
* Eclipse
* Text Editor

**4.2 Software Links**

**Java Developmen Kit:** <https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html>

**Eclipse:** <https://www.eclipse.org/downloads/packages/release/kepler/sr1/eclipse-ide-java-developers>

**Text Editor:**

https://notepad-plus-plus.org/downloads/

**4.3 Hardware Requirements:**

This system can run on any hardware that can handle the above softwares.

**5.0 System Design**

**5.1 Overview:**

The system is divided into 2 parts, Database part containing the list of transactions and the second part is the program where we implement the Apriori Algorithm and after that send the same dataset through Brute Force to see the time difference. We have stored Database in the form of a text file, which is easy to read in Java. The system uses Java inbuilt file handlers to access the file.

**5.2 Database Design:**

The database consists of 30 unique items and we have 20 transactions in each and every database. In total, we have 5 databases in the form of text file.(D1.txt, D2.txt, D3.txt, D4.txt and D5.txt).

On each line we have a new transaction, in which all the items are separated by “,”.

**Text

Description automatically generated**

Transaction example: LedTV,Facemask,Tissues.

**6.0 Database:**

We have 5 Databases, each consisting of 20 transactions.

Our database consists of 30 different items. Database was created in a text file.

**6.1 ItemList**

|  |  |  |
| --- | --- | --- |
| Iphone11 | Airpods | ExtensionCord |
| DoubleSidedTape | Poster | Bike |
| Helmet | BikeCover | AirpodsCover |
| TidePods | Hershey | PlayStation |
| CoffeeMaker | LedTV | Rug |
| DunkinGroundCoffee | CoffeeFilters | Sugar |
| BoseSoundLink | BoseCradle | AABatteries |
| AlexaDot | SmartBulb | BodyWash |
| FaceWash | Loofah | Salt |
| Wallet | Tissues | FaceMask |

**6.2 Transactions in Database 1**

List of Transactions in database 1 is as follows:

**Text

Description automatically generated**

List of transactions in database 2 is as follows:

**Text

Description automatically generated**

List of transactions in Database3

**Text

Description automatically generated**

List of transactions in Database4

**Text

Description automatically generated**

List of Transactions in Database5

**Text

Description automatically generated**

1. **Implementation:**

The system implements Apriori Algorithm and Brute Force method to find associations. It is written in Java using HashMap, HashSet and some other data structures. The system takes 3 input from the user which are File location, Support and Confidence Values.

* 1. **Functions:**

**Combination:**

This function is used to generate combinations.

**Graphical user interface, text, application

Description automatically generated**

**Candidate Rule:Text

Description automatically generated**

**Java.lang.System.currentTimeMillis() Method:**

For comparing the efficiency between Apriori algorithm and Brute Force, we have used a method to calculate the difference between start and end of the program.

**7.2 Basics:**

**HashMap:**

HashMap is a Map based collection class that is used for storing Key & value pairs, it is denoted as HashMap<Key, Value> or HashMap<K, V>. This class makes no guarantees as to the order of the map. It is similar to the Hashtable class except that it is unsynchronized and permits nulls(null values and null key). It is not an ordered collection which means it does not return the keys and values in the same order in which they have been inserted into the HashMap. It does not sort the stored keys and Values. You must need to import java.util.HashMap or its super class in order to use the HashMap class and methods.

**HashSet:**

HashSet extends AbstractSet and implements the Set interface. It creates a collection that uses a hash table for storage.

A hash table stores information by using a mechanism called hashing. In hashing, the informational content of a key is used to determine a unique value, called its hash code.

The hash code is then used as the index at which the data associated with the key is stored. The transformation of the key into its hash code is performed automatically.

**File Input:**

**BufferedReader** is a Java class to reads the text from an Input stream (like a file) by buffering characters that seamlessly reads characters, arrays or lines. In general, each read request made of a Reader causes a corresponding read request to be made of the underlying character or byte stream.

1. **Execution:**

For Execution, we will be asking user to enter Database name, support value and confidence values and running it through Apriori algorithm and Brute Force method.

Ex.

Apriori-----------D1.txt---------- Support----20%, Confidence----50%

BruteForce-----D1.txt-----------Support----20%, Confidence----50%

We will run it through all the 5 databases and compare the running time.

* 1. **Step 1:**

In step 1, we have to enter the location of the file where we have kept the Dataset. While giving the location, we have to take care to enter the extension of file too.

**Graphical user interface, text, application

Description automatically generated**

**8.2 Step 2:**

Here we have to insert the support between 0-100.

**Graphical user interface, text, application, chat or text message

Description automatically generated**

**8.3 Step 3:**

Here we have to insert the confidence between 0-100

**Graphical user interface, text

Description automatically generated**

1. **Testing:**

Testing on Dataset6(D6.txt), which comprises of the following items:

Yogurt, bread, egg, milk, juice and cheese.

Transactions are as follows:

Text

Description automatically generated

Running it on Apriori:

**A picture containing table

Description automatically generated**

**Text

Description automatically generated**

**Items satisfying the association rules are given above with the running time of 3094msec.**

**Running the same dataset with Brute Force.**

**Graphical user interface, text, application, email

Description automatically generated**

**A picture containing text

Description automatically generated**

**A picture containing text

Description automatically generatedText

Description automatically generated**

**A picture containing text

Description automatically generated**

**A picture containing graphical user interface

Description automatically generated**

**Running time with Brute force is 3359msec.**

**9.1 Testing Apriori on D1.txt**

**Source Code for Apriori Algorithm.**

|  |
| --- |
| package Algo**;**  **import** java**.**io**.**BufferedReader**;**  **import** java**.**io**.**File**;**  **import** java**.**io**.**FileReader**;**  **import** java**.**text**.**DecimalFormat**;**  **import** java**.**text**.**NumberFormat**;**  **import** java**.**util**.**Arrays**;**  **import** java**.**util**.**HashMap**;**  **import** java**.**util**.**HashSet**;**  **import** java**.**util**.**Iterator**;**  **import** java**.**util**.**Map**;**  **import** java**.**util**.**Scanner**;**  public class BruteForce **{**  static String temp\_arr**[];**  static float finalSupp**;**  static long start**;**  static long end**;**  public static void main**(**String**[]** args**)** **{**  HashMap**<**HashSet**<**String**>,** Integer**>** Trans **=** **new** HashMap**<>();**  HashMap**<**HashSet**<**String**>,** Integer**>** storage **=** **new** HashMap**<>();**  HashSet**<**String**>** Item **=** **new** HashSet**<>();**  HashSet**<**String**>** ItemAfterStage1 **=** **new** HashSet**<>();**  int support **=** 2**;**    int confidence **=** 2**;**  int totaltrans **=** 0**;**  BufferedReader br**;**  int check **=** 1**;**  start **=** System**.**currentTimeMillis**();**  Scanner myObj **=** **new** Scanner**(**System**.**in**);**  System**.**out**.**println**(**"Please enter the file name:"**);** //Enter a valid file name  String input **=** myObj**.**nextLine**();**  File file **=** **new** File**(**input**);**  **try** **{**    //Scanner myObj = new Scanner(System.in);  System**.**out**.**println**(**"Enter Support Value in percentage Eg:70 for 70%"**);** // enter support between 0-100  support **=** myObj**.**nextInt**();**  //support = 10;  finalSupp**=**20**;**  System**.**out**.**println**(**"Enter Confidence Value in percentage Eg:70 for 70%"**);** //enter confidence between 0-100  confidence **=** myObj**.**nextInt**();**  //confidence = 15;  //myObj.close();  br **=** **new** BufferedReader**(new** FileReader**(**file**));** //Start reading the file from here  String st**;**  **while** **((**st **=** br**.**readLine**())** **!=** **null)** **{**  String strArray**[]** **=** st**.**split**(**","**);**  HashSet**<**String**>** setOfTrans **=** **new** HashSet**<>(**Arrays**.**asList**(**strArray**));**  **if** **(**Trans**.**containsKey**(**setOfTrans**))** **{**  Trans**.**put**(**setOfTrans**,** Trans**.**get**(**setOfTrans**)** **+** 1**);**  **}** **else** **{**  Trans**.**put**(**setOfTrans**,** 1**);**  **}**  totaltrans**++;**  Item**.**addAll**(**setOfTrans**);**  **}**  //System.out.println(Trans);// HashMap containing all the lines of the file.  //System.out.println(Item);  support **=** **(**int**)** **(((**float**)** support **/** 100**)** **\*** totaltrans**);**  Iterator**<**String**>** value **=** Item**.**iterator**();**  **while** **(**value**.**hasNext**())** **{**  String itemTemp **=** value**.**next**();**  int val **=** 0**;**  **for** **(**Map**.**Entry**<**HashSet**<**String**>,** Integer**>** entry **:** Trans**.**entrySet**())** **{**  **if** **(**entry**.**getKey**().**contains**(**itemTemp**))** **{**  val **+=** entry**.**getValue**();**  **}**  **}**  // if (val >= support) {  ItemAfterStage1**.**add**(**itemTemp**);**  HashSet**<**String**>** tempStorage **=** **new** HashSet**<>();**  tempStorage**.**add**(**itemTemp**);**  storage**.**put**(**tempStorage**,** val**);**  //}  **}**  //System.out.println(storage);// the count after the first iteration  System**.**out**.**println**(**"--------------------------------"**);**  System**.**out**.**println**(**"The individual items and their count"**);**  **for** **(**HashSet**<**String**>** str **:** storage**.**keySet**())** **{**  System**.**out**.**println**(**str **+** " = " **+** storage**.**get**(**str**));**  **}**  System**.**out**.**println**(**"--------------------------------"**);**  temp\_arr **=** ItemAfterStage1**.**toArray**(new** String**[**0**]);**  //System.out.println("= " + Arrays.toString(temp\_arr));  int exit **=** 0**;**  **while** **(**exit **==** 0**)** **{**  exit **=** 1**;**  setIteration**();**  obj **=** **new** HashSet**<>();**  combination**(new** String**[**iteration**],** 0**,** 0**);**  System**.**out**.**println**(**"-------------------------------"**);**  System**.**out**.**println**(**"Item list after the iteration : " **+** getIteration**());**  **for** **(**HashSet**<**String**>** comb **:** obj**)** **{**  System**.**out**.**println**(**comb**);**  **}**  System**.**out**.**println**(**"-------------------------------"**);**  **for** **(**HashSet**<**String**>** comb **:** obj**)** **{**  int k **=** 0**;**  **if** **(**k **==** 0**)** **{**  int val **=** 0**;**  **for** **(**Map**.**Entry**<**HashSet**<**String**>,** Integer**>** entry **:** Trans**.**entrySet**())** **{**  **if** **(**entry**.**getKey**().**containsAll**(**comb**))** **{**  val **+=** entry**.**getValue**();**  **}**  **}**  storage**.**put**(**comb**,** val**);**  **if** **(**val **>=** support**)** **{**    exit **=** 0**;**  **}**  **}**  **}**    System**.**out**.**println**(**"The individual items and their count after the iteration :" **+**  getIteration**());** **for** **(**HashSet**<**String**>** str **:** storage**.**keySet**())** **{**  System**.**out**.**println**(**str **+** " = " **+** storage**.**get**(**str**));** **}**  System**.**out**.**println**(**"--------------------------------"**);**    **}**  System**.**out**.**println**(**"The final frequent item list is :"**);**  **for** **(**HashSet**<**String**>** frr **:** storage**.**keySet**())** **{**  System**.**out**.**println**(**frr **+** "-->" **+** storage**.**get**(**frr**));**  **}**  System**.**out**.**println**(**"-------------------------------------------------"**);**  System**.**out**.**println**(**"The items satisfying the association rules are :"**);**  **for** **(**Map**.**Entry**<**HashSet**<**String**>,** Integer**>** entry **:** storage**.**entrySet**())** **{**  atest **=** **new** String**[**entry**.**getKey**().**size**()];**  entry**.**getKey**().**toArray**(**atest**);**  smt **=** atest**;**  **for** **(**int u **=** 1**;** u **<** atest**.**length**;** u**++)** **{**  candidateRule**(new** String**[**u**],** 0**,** 0**,** u**);**  **}**  HashSet**<**String**>** tempx **=** **new** HashSet**<>();**  tempx**.**addAll**(**entry**.**getKey**());**  **for** **(**HashSet**<**String**>** ls **:** orr**)** **{**  tempx**.**removeAll**(**ls**);**  NumberFormat formatter **=** **new** DecimalFormat**(**"#0.00"**);**  Integer t2 **=** entry**.**getValue**();**  Integer t3 **=** 0**;**  **if** **(**storage**.**containsKey**(**ls**))** **{**  t3 **=** storage**.**get**(**ls**);**  **}** **else** **{**  System**.**out**.**println**(**"error"**);**  **}**  float conf **=** **((**float**)** t2 **/** t3**)** **\*** 100**;**  float supp **=** **((**float**)** t2 **/** totaltrans**)** **\*** 100**;**  //System.out.println(supp+"="+finalSupp);  **if** **(**conf **>=** confidence **&&** supp**>=**finalSupp**)** **{**  check **=** 0**;**  System**.**out**.**println**(**ls**.**toString**().**replace**(**"["**,** ""**).**replace**(**"]"**,** ""**)** **+** "-->"  **+** tempx**.**toString**().**replace**(**"["**,** ""**).**replace**(**"]"**,** ""**)** **+** ":[" **+** formatter**.**format**(**supp**)**  **+** "%," **+** formatter**.**format**(**conf**)** **+** "%]"**);**  **}**  tempx**.**addAll**(**entry**.**getKey**());**  **}**  orr **=** **new** HashSet**<**HashSet**<**String**>>();**  **}**  // END Calculating the support and confidence  **if** **(**check **==** 1**)** **{**  System**.**out**.**println**(**"Nothing to display with the given confidece and support"**);**  **}**  **}** **catch** **(**Exception e**)** **{**  e**.**printStackTrace**();**  **}**  end **=** System**.**currentTimeMillis**();**  System**.**out**.**println**(**"Total time to run the program is : "**+(**end **-**start**));**  **}**  static String**[]** smt **=** **{};**  static HashSet**<**HashSet**<**String**>>** obj **=** **new** HashSet**<>();**  static int iteration **=** 1**;**  static String**[]** atest**;**  static HashSet**<**HashSet**<**String**>>** orr **=** **new** HashSet**<**HashSet**<**String**>>();**  public static int getIteration**()** **{**  **return** iteration**;**  **}**  public static void setIteration**()** **{**  BruteForce**.**iteration **+=** 1**;**  **}**  public static void combination**(**String data**[],** int start**,** int index**)** **{**  **if** **(**index **==** iteration**)** **{**  HashSet**<**String**>** temp **=** **new** HashSet**<>();**  **for** **(**int j **=** 0**;** j **<** iteration**;** j**++)**  temp**.**add**(**data**[**j**]);**  obj**.**add**(**temp**);**  **return;**  **}**  **for** **(**int i **=** start**;** i **<=** temp\_arr**.**length **-** 1 **&&** temp\_arr**.**length **-** i **>=** iteration **-** index**;** i**++)** **{**  data**[**index**]** **=** temp\_arr**[**i**];**  combination**(**data**,** i **+** 1**,** index **+** 1**);**  **}**  **}**  public static void candidateRule**(**String data**[],** int start**,** int index**,** int iterator**)** **{**  **if** **(**index **==** iterator**)** **{**  HashSet**<**String**>** temp **=** **new** HashSet**<>();**  **for** **(**int j **=** 0**;** j **<** iterator**;** j**++)**  temp**.**add**(**data**[**j**]);**  orr**.**add**(**temp**);**  **return;**  **}**  **for** **(**int i **=** start**;** i **<=** smt**.**length **-** 1 **&&** smt**.**length **-** i **>=** iterator **-** index**;** i**++)** **{**  data**[**index**]** **=** smt**[**i**];**  candidateRule**(**data**,** i **+** 1**,** index **+** 1**,** iterator**);**  **}**  **}**  **}** |

**Conclusion:**

In this project, datasets were set through Apriori and Brute force with same set of constrains. After seeing the running time in each and every case, we can conclude that Apriori is efficient in each and every test case irrespective of the size of the dataset. The system also generated frequent itemsets.

By using HashMaps and HashSets, fetch operations took only O(1) time. It reduces the size of the itemsets in the database considerably providing a good performance. Thus, data mining helps consumers and industries better in the decision-making process.

**References**

1. [**https://www.hackerearth.com/blog/developers/beginners-tutorial-apriori-algorithm-data-mining-r-implementation/#:~:text=Apriori%20algorithm%20is%20a%20classical,by%20customers%20in%20a%20store**](https://www.hackerearth.com/blog/developers/beginners-tutorial-apriori-algorithm-data-mining-r-implementation/#:~:text=Apriori%20algorithm%20is%20a%20classical,by%20customers%20in%20a%20store)**.**
2. <https://beginnersbook.com/2013/12/hashmap-in-java-with-example/>
3. <https://www.tutorialspoint.com/java/lang/system_currenttimemillis.htm>
4. <https://www.tutorialspoint.com/java/java_hashset_class.htm>