MANIPAL UNIVERSITY

DEPARTMENT OF INFORMATION & COMMUNICATION TECHNOLOGY

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL

CERTIFICATE

This is to certify that Ms./Mr. ……...…… Reg.No. …..…… Section: …… Roll No: …..…… has satisfactorily completed the lab exercises prescribed for Data Mining and Predictive Analysis Lab [ICT 3262] of Sixth Semester B. Tech. (CCE) Degree at MIT, Manipal, in the academic year Jan-May2019.

Date: …….....

Signature of the faculty

CONTENTS

Course Objectives

- Implementation of frequent pattern finding algorithms for association rule mining
- Implementation of clustering algorithms
- Implementation of the classification algorithms.
- Implementation of a mini project on application of the above.
- Familiarization with Rapid miner and InfoSphere

Course Outcomes

At the end of this course, students will be able to

- Perform pre-processing on the given raw data set using data mining software tool.
- Able to identify and apply the suitable data mining technique on pre-processed data.
- Improves analytic skills and problem solving skills.

Evaluation plan

Split up of 60 marks for Regular Lab Evaluation
Lab 1 to Lab 4: (20 Marks) [Record:8M Execution:4M Viva/Execution Test:8M] Lab 5 to 6 : (10 Marks) [Record:4M Execution:2M Viva/Execution Test:4M] Lab 7 to Lab 10: (30 Marks)[Record:12M Execution:6M Viva/Execution Test:12M]
End Semester Lab evaluation: 40 marks (Duration 2 hrs)
Miniproject : 20 Marks Endsem: 20Marks

INSTRUCTIONS TO THE STUDENTS

Pre- Lab Session Instructions

1. Students should carry the Lab Manual Book and the required stationery to every lab session
2. Be in time and follow the institution dress code
3. Must sign in the log register provided
4. Make sure to occupy the allotted seat and answer the attendance
5. Adhere to the rules and maintain the decorum

In- Lab Session Instructions

- Follow the instructions on the allotted exercises
- Show the program and results to the instructors on completion of experiments
- Prescribed textbooks and class notes can be kept ready for reference if required

General Instructions for the exercises in Lab

- Implement the given exercise individually and not in a group.
- The programs should meet the following criteria:
    - Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs.
    - Comments should be used to give the statement of the problem.
    - Statements within the program should be properly indented.
- Plagiarism (copying from others) is strictly prohibited and would invite severe penalty in evaluation.
- In case a student misses a lab, he/ she must ensure that the experiment is completed before the next evaluation with the permission of the faculty concerned.
- Students missing out lab on genuine reasons like conference, sports or activities assigned by the Department or Institute will have to take **prior permission** from the HOD to attend **additional lab** (with other batch) and complete it **before** the student goes on leave. The student could be awarded marks for the write up for that day provided he submits it during the **immediate** next lab.

- Students who fall sick should get permission from the HOD for evaluating the lab records. However attendance will not be given for that lab.
- Students will be evaluated only by the faculty with whom they are registered even though they carry out additional experiments in other batch.
- Presence of the student during the lab end semester exams is mandatory even if the student assumes he has scored enough to pass the examination
- Minimum attendance of 75% is mandatory to write the final exam.
- If the student loses his book, he/she will have to rewrite all the lab details in the lab record.
- Questions for lab tests and examination are not necessarily limited to the questions in the manual, but may involve some variations and / or combinations of the questions.

THE STUDENTS SHOULD NOT

- Bring mobile phones or any other electronic gadgets to the lab.
- Go out of the lab without permission.

LAB NO: 1 **Date:**

RAPID MINER OPERATORS FOR DATA ACCESS

Objectives:

- **To understand the different operators for data access in Rapid miner.**

Introduction:

Rapid Miner Studio is open source and in process view of the Studio there are five panels. They are Repository, Operators, Process, Parameters and Help. Figure 1 shows the overall panels in process view of the tool. Rapid Miner Studio provides operators for Data Access, Blending, Cleansing, Modeling, Scoring, Validation and Utility.

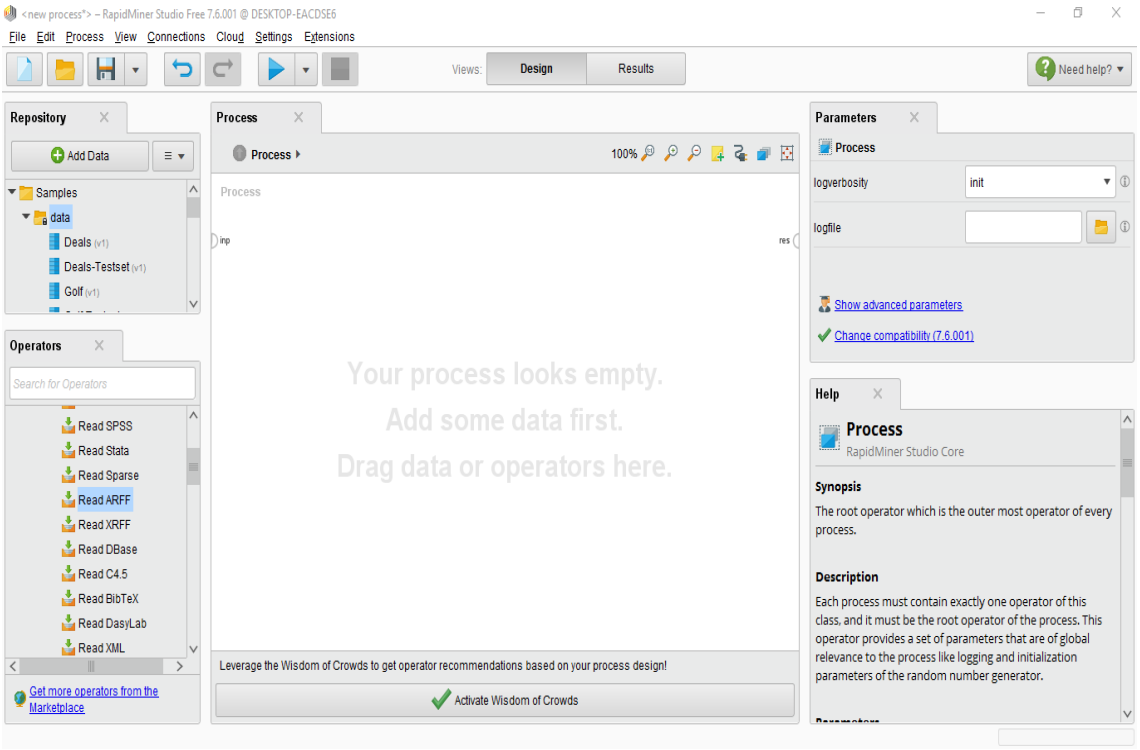


Figure 1. Process View

This section provides details about various operators in Rapid Miner Studio such as Data Access operators.

1. Data Access Operators

This section has operators to

i) read and write data from and into File respectively.
ii) read, write and update Database
iii) obtain data from Web Applications

Reading from Files

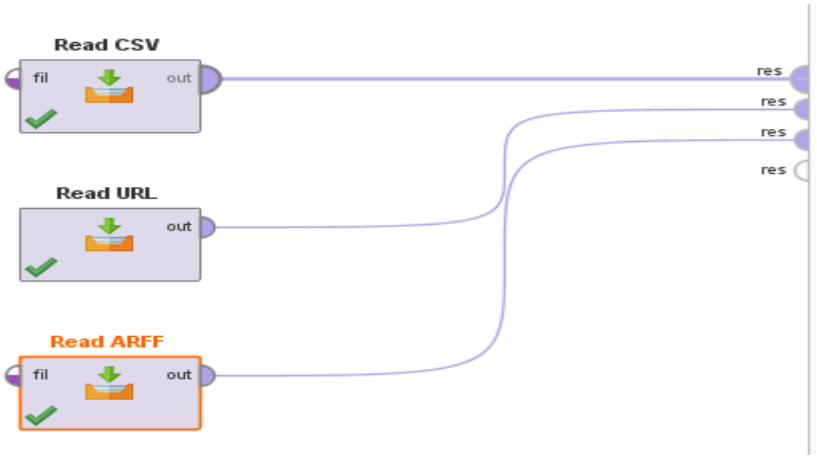


Figure 2. Example operators for reading from file

As shown in Figure 2 drag and drop Read CSV, Read URL and Read ARFF operators to the process panel and connect their out ports to res port. Set the configuration for each operator in parameter panel by clicking on respective operator icon. In order to see the result you have to click on Execute button in top panel.

Read-Write operation

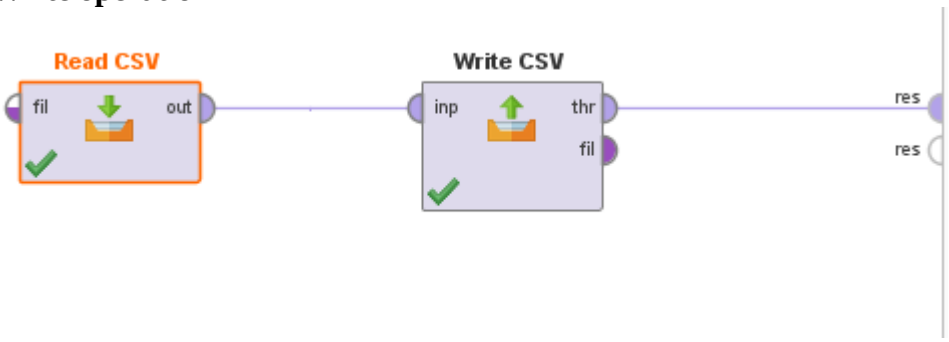


Figure 3. Read Write Operation

As shown in Figure 3, in order to write into a CSV file drag and drop a Read CSV and Write CSV operators to the Process Panel and then connect out port of Read CSV operator icon to inp port of Write CSV operator. The thr port of Write CSV should be connected to res port. The path of the file into which content has to be written should be given in parameter panel. Using repeated Write operator, you can write into multiple files. Rapid miner also provides sample data set whose operators can be dragged and dropped on to process panel. Table 1 provides various port abbreviations and their meaning and description in each operator icon.

Table 1. Port Abbreviations

Port Abbreviation	Meaning	Description
ass	*Association*	Association rules that have been discovered in a frequent item set
att	*Attribute*	Attribute weights (in and out)
ave	*Average*	Performance measures; estimate of performance using the model built on the complete delivered data set
clu	*Cluster model*	Cluster model created when clustering an example set
exa	*Example set*	Example set
fil	*File*	File object
for	*Formula*	Formula result
fre	*Frequent*	Frequent item or item sets for association rule learning
gro	*Grouped*	Grouped models, attributes, items
hie	*Hierarchical*	Hierarchical clustering model
inp	*Input*	Input source, can take various objects

ite	*Item sets*	Frequent item sets (groups of items that often appear together in the data)
joi	*Join*	Join of the left and right example sets
lab	*Labeled data*	Model that was given in input is applied on the example set and the updated example set is delivered from this port
lef	*Left*	Left input port expecting an example set, which is used as the left example set for a join
mat	*Matrix*	Correlations matrix of all attributes of the input example set
mer	*Merged*	Merged example set
mod	*Model*	Default model from this output port
obj	*Object*	IO object
ori	*Original*	Input example set is passed without changing to this port
out	*Output*	Output port

Adding Twitter data

Drag and drop the twitter icon on process panel. Connect the out port to res port.

Figure 4. Adding Twitter data

In the parameter window, do the following to set the connection:

1. Create an access token for Rapid Miner to request twitter feed.

2. Click on twitter symbol against connection field.

3. Click Add Connection - Give a connection name and select connection type as twitter connection from the drop down. Click on create button.

4. Against access token field, paste the access token received (can be created in https://apps.twitter.com/app/) and click on button, Request access token as shown in the screenshot below.

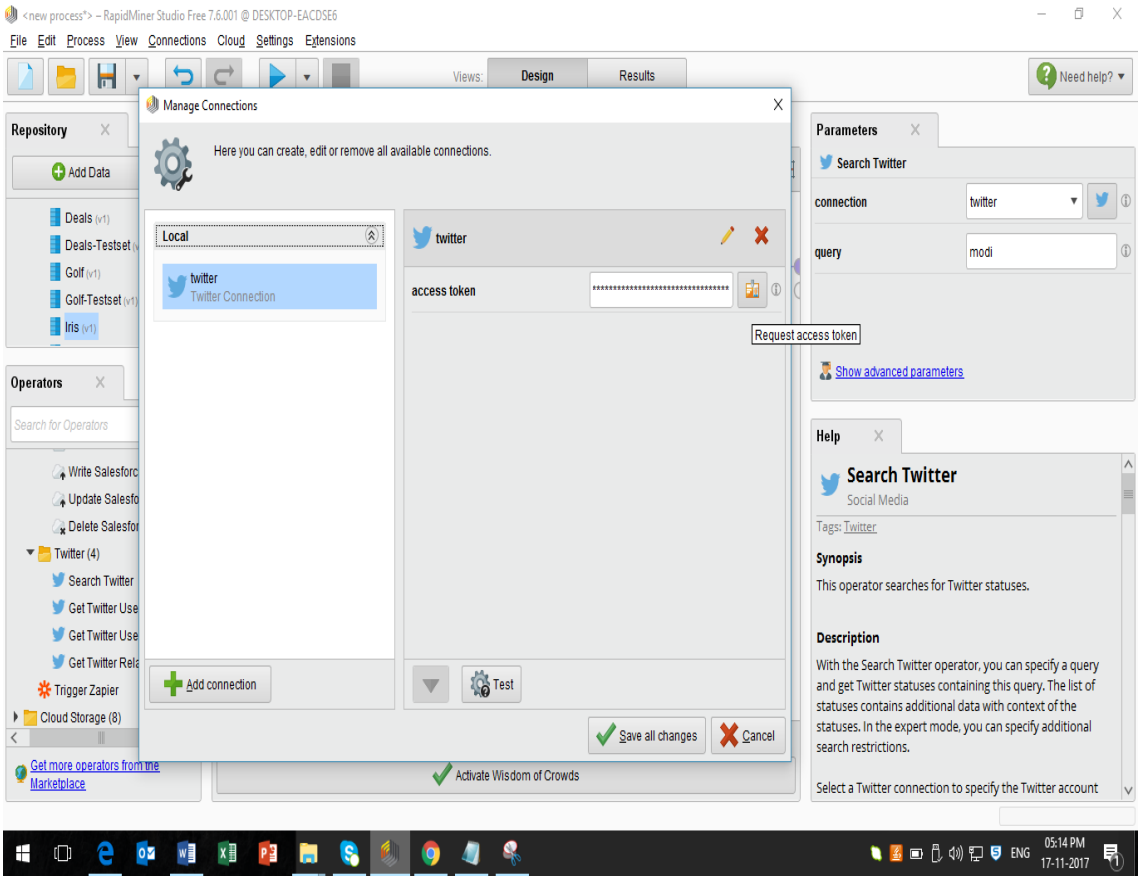


Figure 5. Setting access token

5. On click of the Request access token button the following window will appear

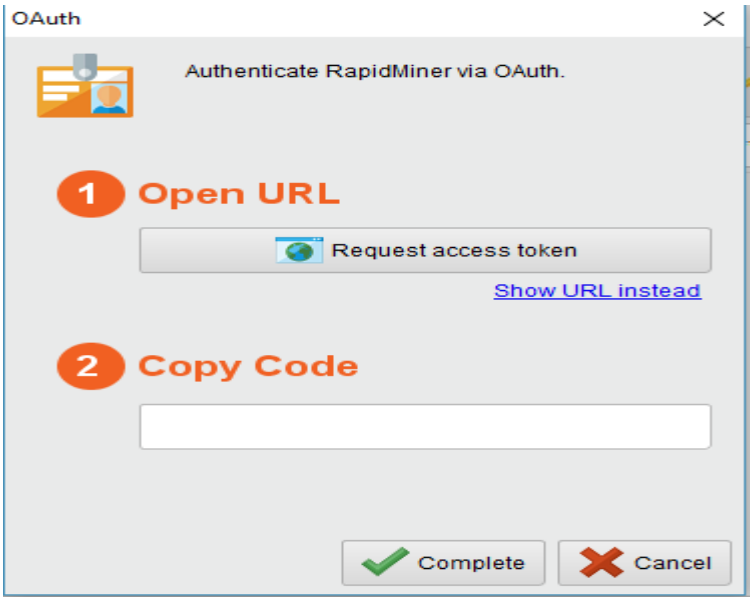


Figure 6. Authentication

6. Click on Request access token the following web page will appear

Figure 7. Authorization

7. Click on Authorize app.

8. You will receive a PIN which needs to be entered in “COPY CODE” field. Click on complete.

9. Click on Test and followed by Save all changes.

10. Enter a query in query field. Example “Modi”

11. Execute. The result will be as follows.

	Id	Created-At	From-User	From-User-Id	To-User	To-User-Id	Language	Source	Text
	9309942641...	Nov 16, 2017 ...	Office of RG	3171712086	?	-1	en	<a href="http:/...	Modi ji - nice t...
	9310547262...	Nov 16, 2017 ...	Amit Shah	1447949844	?	-1	en	<a href="http:/...	The findings ...
	9310835361...	Nov 16, 2017 ...	BJP	207809313	?	-1	en	<a href="http:/...	Yet another e...
	9314834123...	Nov 17, 2017 ...	Shivanna Ma...	9271721907...	INCIndia	1153045459	in	<a href="http:/...	@INCIndia M...
	9314834102...	Nov 17, 2017 ...	Ankit Srivasta...	70166896	?	-1	en	<a href="http:/...	RT @sharma...
	9314834057...	Nov 17, 2017 ...	smitha060444	9049645668...	?	-1	en	<a href="http:/...	RT @Paperb...
	9314834043...	Nov 17, 2017 ...	Raviraj Virkar	2402557338	?	-1	en	<a href="http:/...	RT @muglika...
	9314833977...	Nov 17, 2017 ...	GLOBALCITI...	621030151	Knkinjal_	9106516548...	in	<a href="http:/...	@Knkinjal_ ...
	9314833971...	Nov 17, 2017 ...	Rajesh Kumar	8963371510...	awasthis	71815077	hi	<a href="http:/...	@awasthis ...
	9314833664...	Nov 17, 2017 ...	Shrimant Mane	120393444	?	-1	en	<a href="http:/...	Moody's Upgr...
	9314833656...	Nov 17, 2017 ...	Sheikh Muha...	8590654530...	?	-1	en	<a href="http:/...	RT @Jagrati...
	9314833630...	Nov 17, 2017 ...	Capt.Nilesh	4188258740	?	-1	en	<a href="http:/...	RT @ggiittiikk...
	9314833618...	Nov 17, 2017 ...	True Fight	3092961320	?	-1	en	<a href="http:/...	RT @rahulro...
	9314833590...	Nov 17, 2017 ...	Retweet Wal...	1366646221	?	-1	en	<a href="http:/...	RT @rahulro...
	9314833577...	Nov 17, 2017 ...	Al Humbert	305060948	?	-1	en	<a href="http:/...	RT @AmarA...

Figure 8. Result of Twitter data access

Database connection in Rapid Miner

1. Drag and drop the Read Database operator on the process panel as shown in Figure 9.

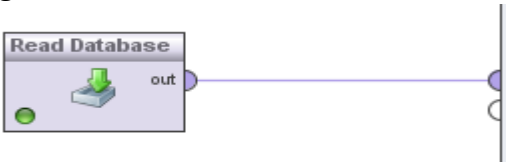


Figure 9. Data base access

2. Do the following configuration in Property window:
    a) Set the define connection to “predefined”.
    b) Click on the button “create, edit, delete database connections “ beside connection as shown in the screenshot below in Figure 10.

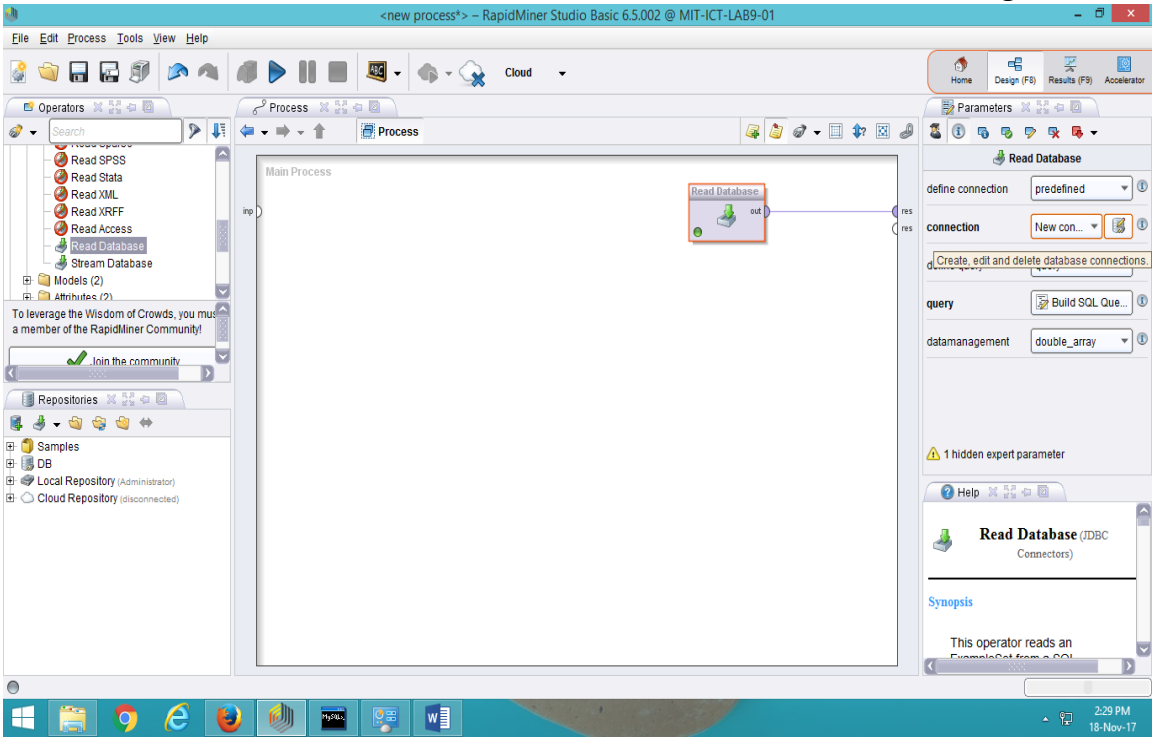


Figure 10. Screen shot for setting parameters for DB connection

    c) A window will appear in which you need to fill up the following information

- Provide a connection name
- Database system: MySql
- Host: Localhost, Port:3306
- Database scheme (The database name you have created in the backend)
- User:root
- Password:student

The Screenshot of the same is given below in Figure 11.

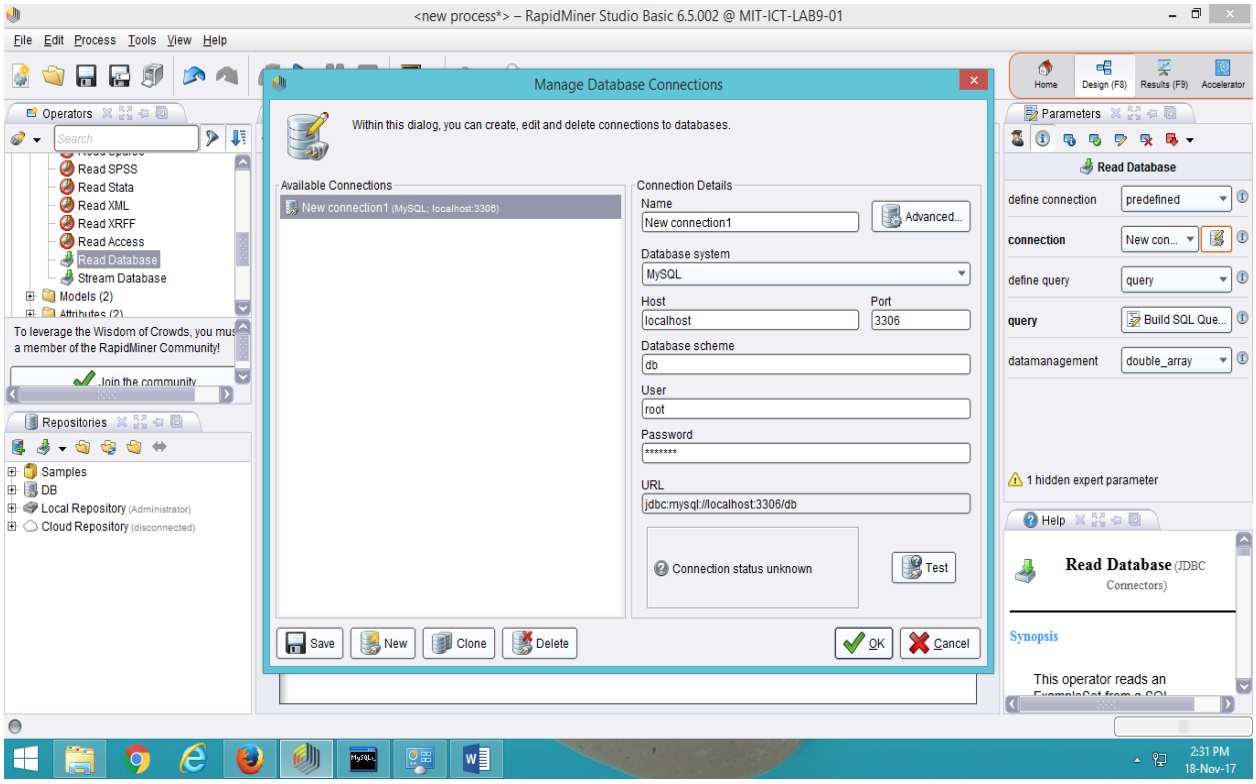


Figure 11. Setting parameters for DB access

d) Click on “Test” button.

3. Click on the query button in the property window. The following window will appear where you need to provide the query as shown below in Figure 12.

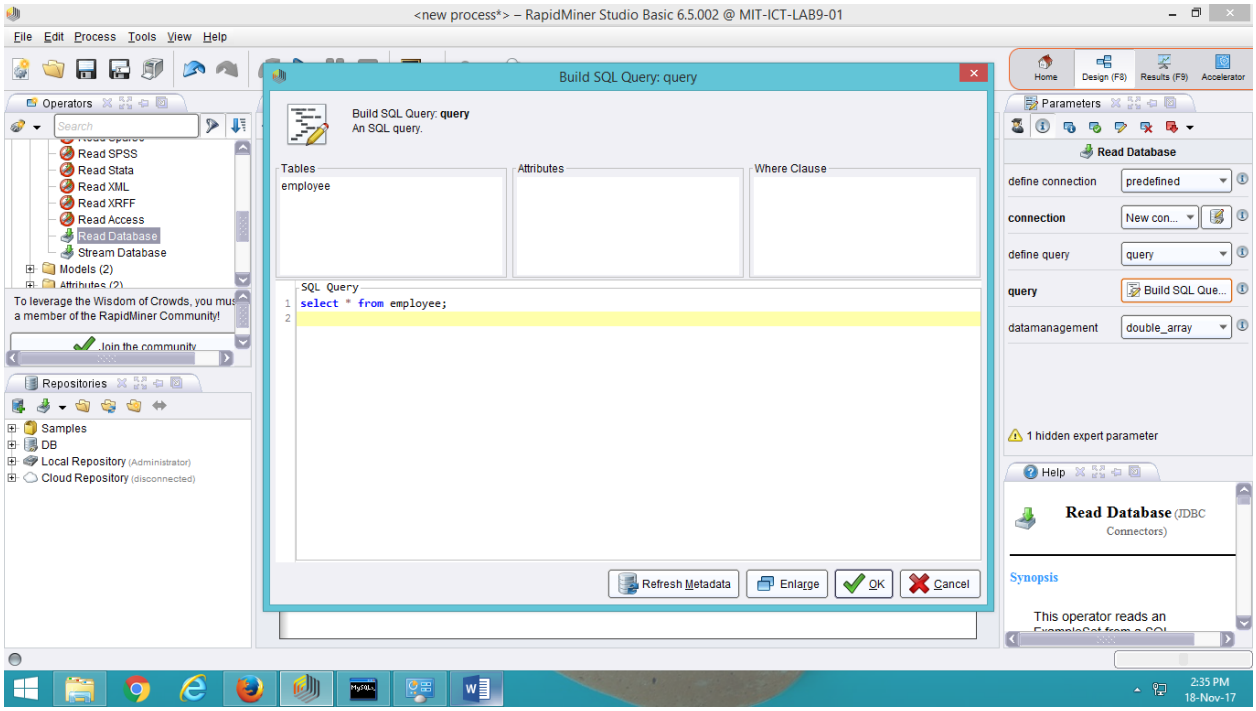


Figure 12. Entering SQL query for DB Access

4. Execute the process and you will get the data asked in the query.

Lab exercises:

1. Explore all the other operators listed under operator panel for data access. For each operator draw the process diagram and list configuration parameters.

LAB NO: 2 **Date:**

RAPID MINER OPERATORS FOR PREPROCESSING

Objectives:

- **To understand the different operators for data preprocessing in Rapid miner.**

1. Blending Operators

I. Attributes

i. Names & Roles

- **Rename: Renames the attribute names.**

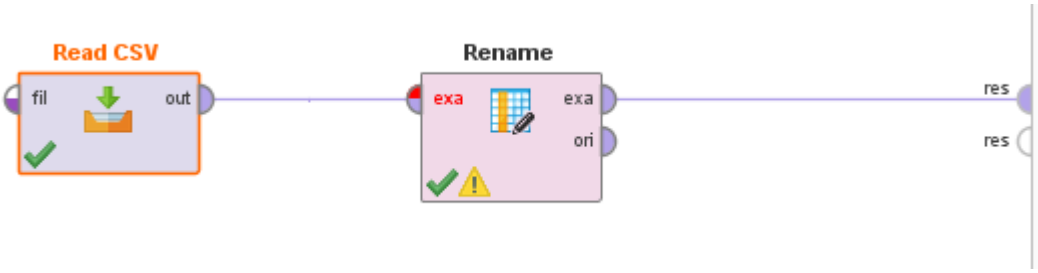


Figure 1. Renaming file

In the parameter window of Rename operator specify the old attribute name in dataset file and the new name which you would like to rename it to.

- **Rename by Generic Names**: This operator renames the selected attributes of the given ExampleSet to a set of generic names like att1, att2, att3 etc.

  The Write Constructions operator writes all attributes of the ExampleSet given at the input port into the specified file. The path of the file is specified through the *attribute constructions file* parameter. Each line in the file holds the construction description of one attribute. The Read Constructions operator can be used for reading this file.

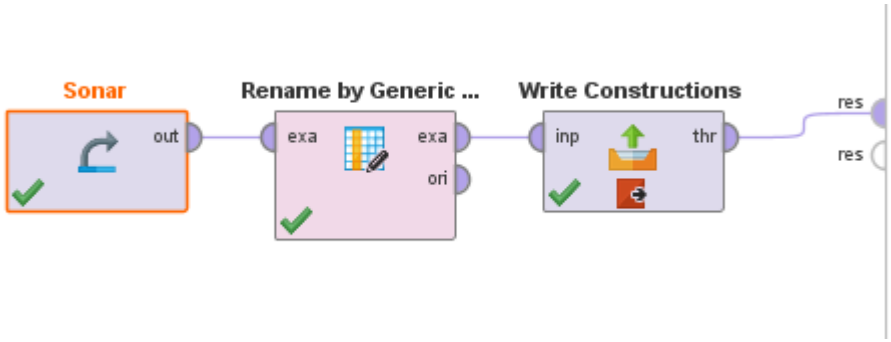


Figure 2. Rename by generic names

ii. **Types:** This operator changes the type of the selected numeric attributes to a binominal type. It also maps all values of these attributes to corresponding binominal values.

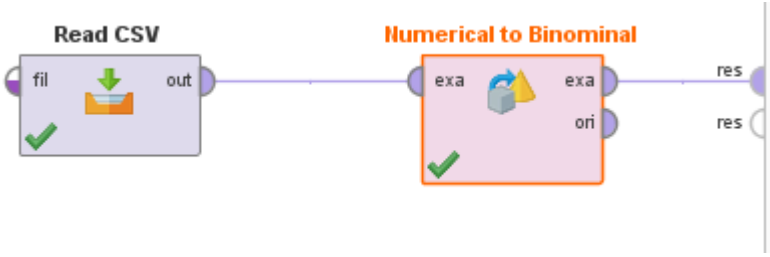


Figure 3. Types

iii. Selection

Select Attributes: This Operator selects a subset of Attributes of an ExampleSet and removes the other Attributes. The following screenshot demos that a subset of attribute is selected in the parameter window and attributes which need to appear is selected in the "Select Attribute button" against "attributes" field.

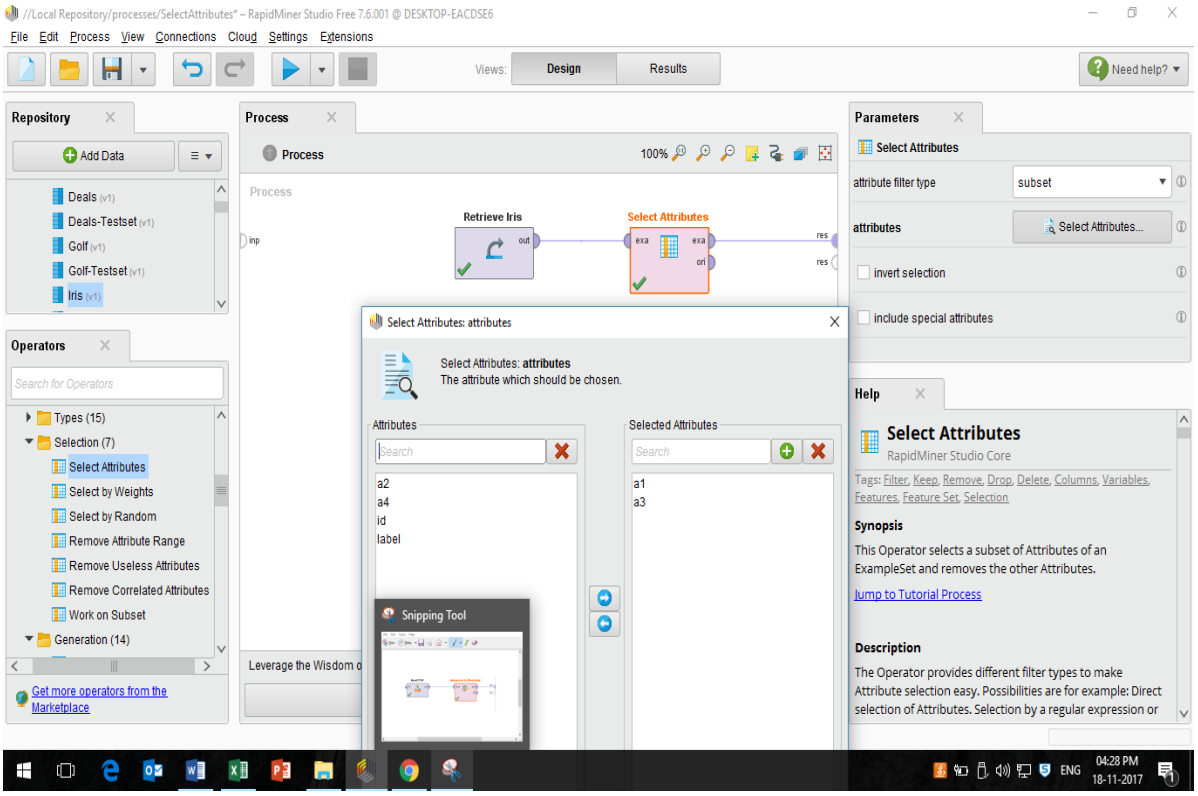


Figure 4. Setting parameters for selection

Remove correlated attributes: This operator removes correlated attributes from an ExampleSet. The correlation threshold is specified by the user in the correlation field of parameter window.

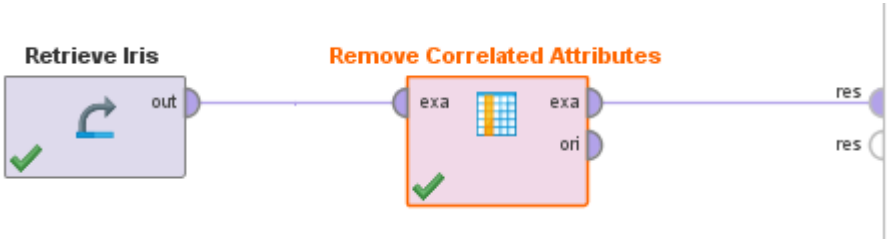


Figure 5. Removing correlated attributes

iv.Genertation

tf-idf: This operator performs a TF-IDF filtering of the given ExampleSet.

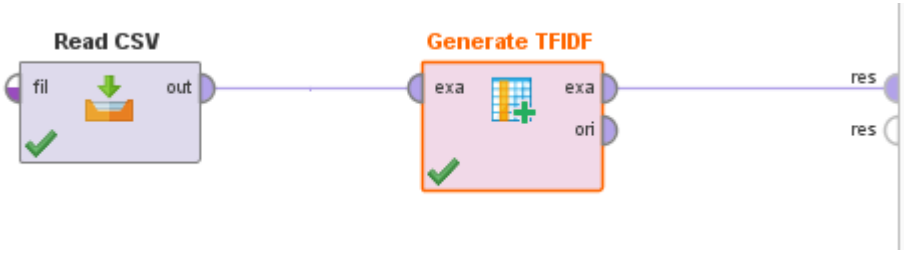


Figure 6. Generation of TFIDF

II. Examples

i)Filter Example operator

Filters examples based on a condition applied on an attribute.

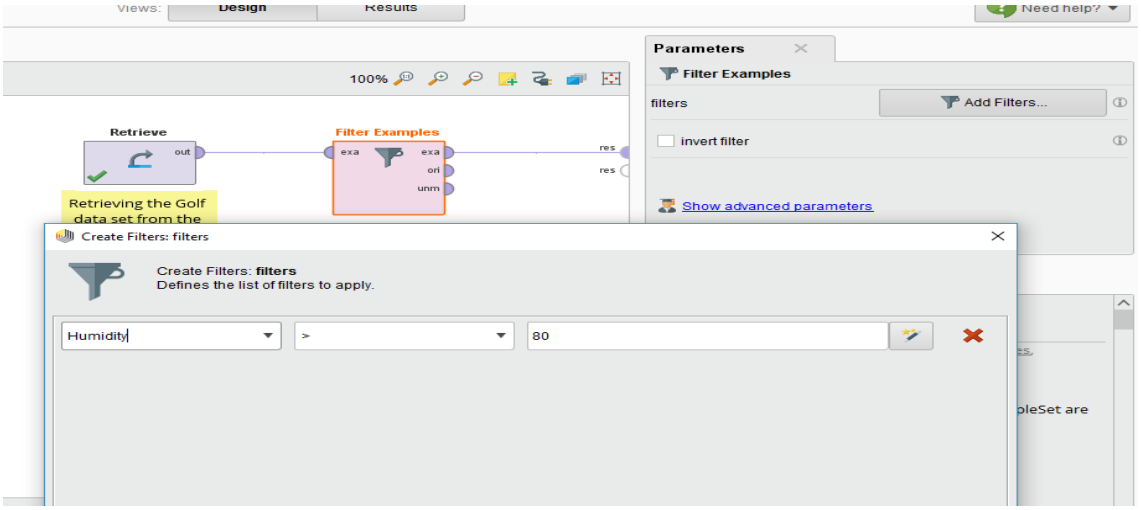


Figure 7. Filter operator parameter setting

ii) Sampling operator

split data:

Splitting data set according to the ratio provided in the parameter panel.

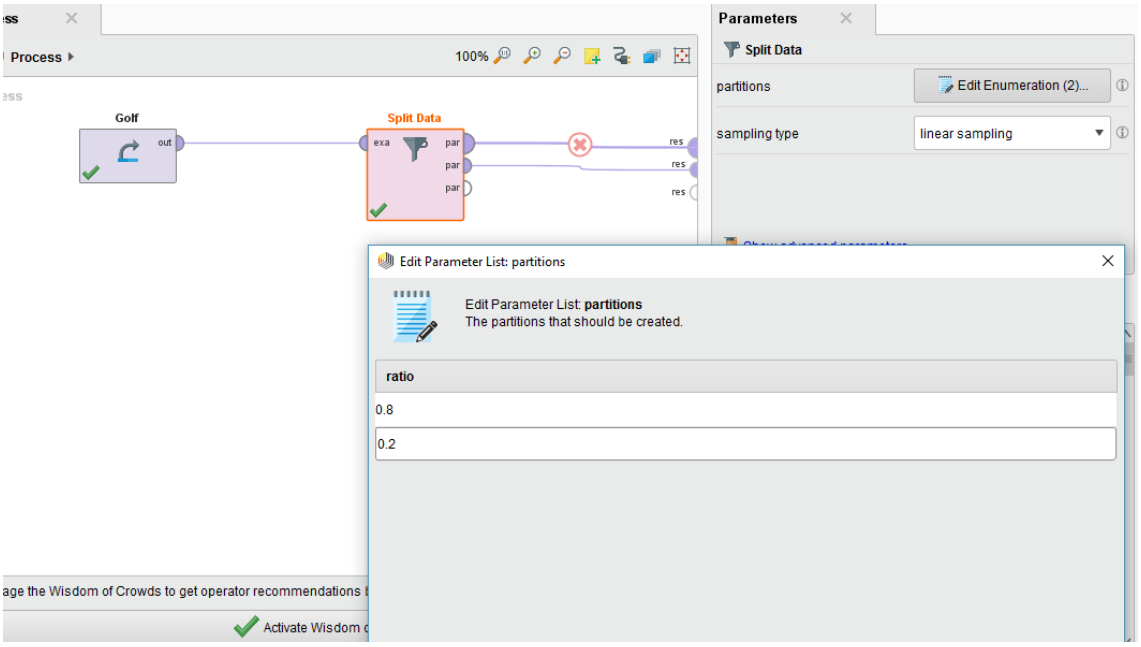


Figure 8. Sampling input data set.

iii)Sorting

Figure 9. Sorting

III. Tables

i)Rotation

Transpose:

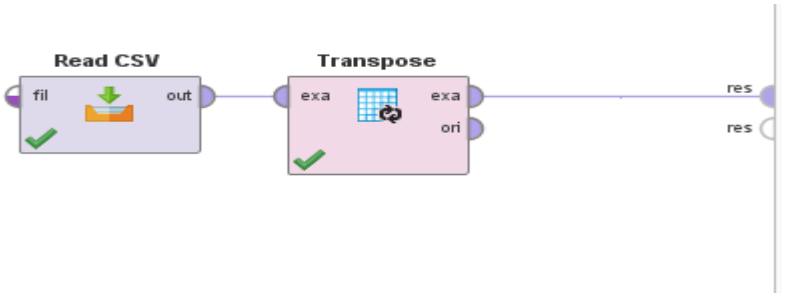


Figure 10. Finding Transpose

ii)Joins

Setminus operator:

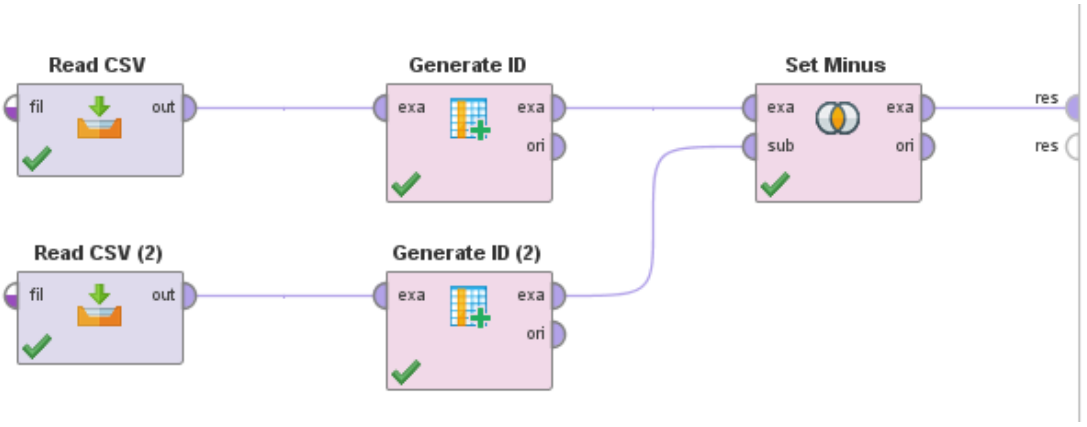


Figure 11. Subtraction of Data set.

2. Data Cleansing Operators:

i)Normalize: This operator normalizes the attribute values of the selected attributes.

ii)Replace Missing Values: This operator replaces missing values in examples of selected attributes by a specified replacement.

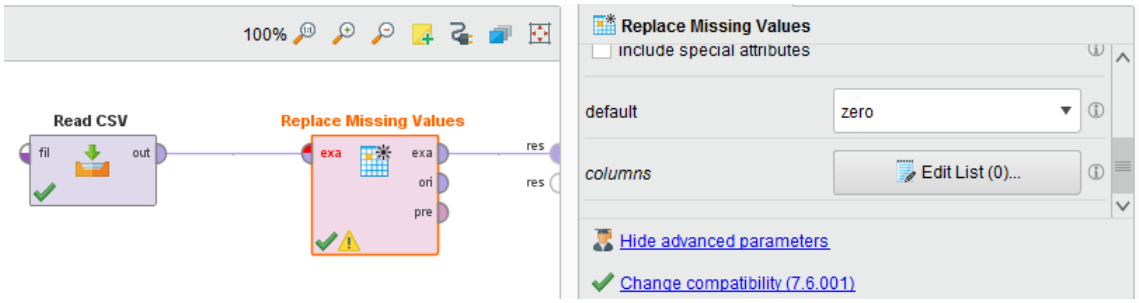


Figure 12. Replacing missing values

iii)Remove Duplicates: This operator removes duplicate examples from an ExampleSet by comparing all examples with each other on the basis of the specified attributes. Two examples are considered duplicate if the selected attributes have the same values in them.

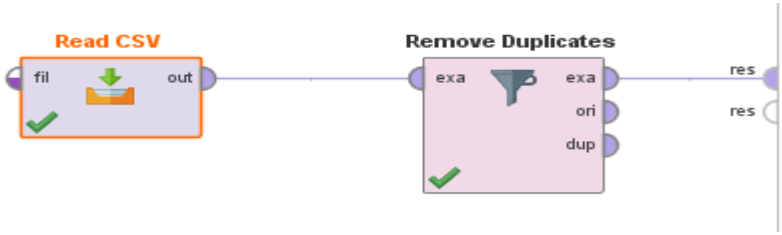


Figure 13. Removing duplicates

iv)Detect Outlier : This operator identifies *n* outliers in the given ExampleSet based on the distance to their *k* nearest neighbors. The variables *n* and *k* can be specified through parameters.

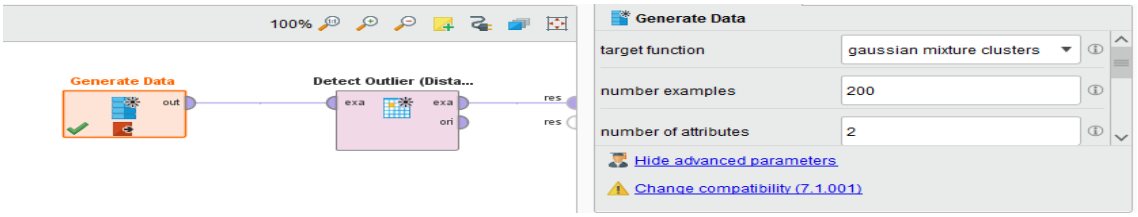


Figure 14. Detection of outliers

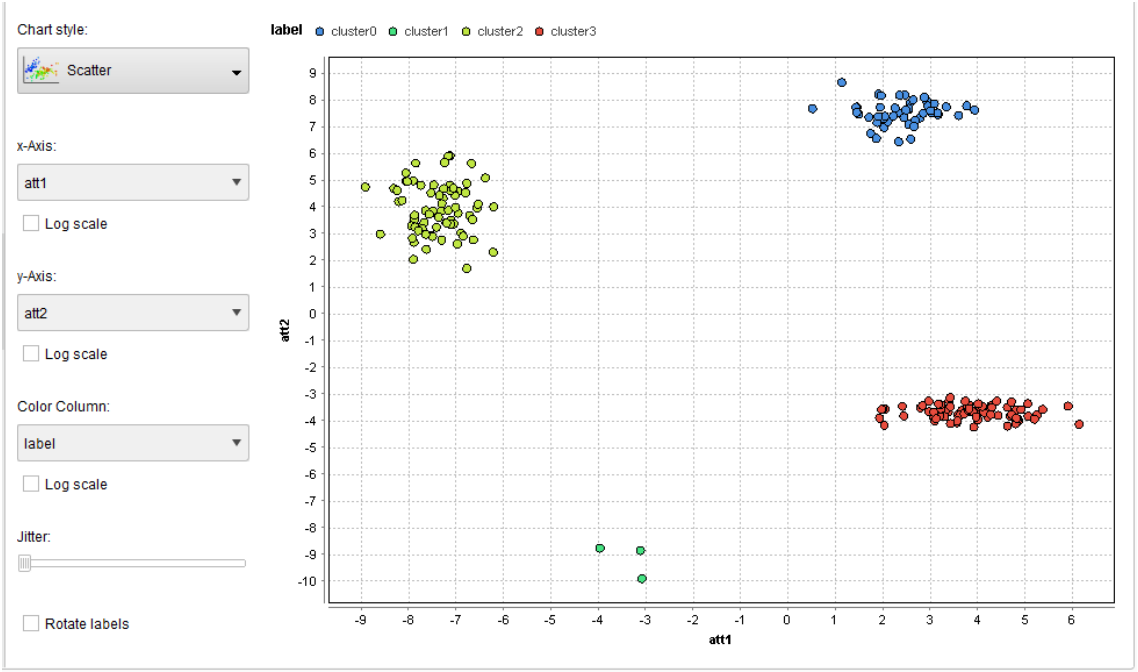


Figure 15. Result of outlier detection

v)Dimesionality Reduction(PCA) : This operator performs a Principal Component Analysis (PCA) using the covariance matrix. The user can specify the amount of variance to cover in the original data while retaining the best number of principal components. The user can also specify manually the number of principal components.

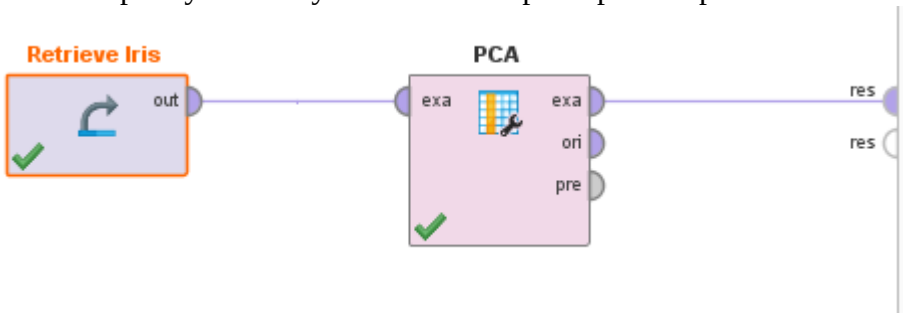


Figure 16. Dimensionality Reduction

Lab exercises:

1. Explore all the other operators listed under operator panel for preprocessing of data. For each operator draw the process diagram and list configuration parameters.

2. Consider a data set with attributes RollNo, Score1, Score2. Perform the following operations.

- Select all attributes except RollNo and display the data.
- How many missing values are there in each attribute?
- Find statistics such as minimum and maximum in Score1 and Score2.
- What is Z-transformation? Apply Z-transformation Score1 and Score2 and find the minimum and maximum value after transformation. Also find the mean and standard deviation.
- Check whether the distribution remains same for Score 1 and Score 2 after applying z-transformation
- What is Range transformation? Apply range transformation and find minimum, maximum and average values.
- Consider the dataset https://www2.stetson.edu/~jrasp/data/AMZN-KO.xls Which is daily returns for the stocks of two companies, Check whether the attributes are highly correlated.

LAB NO: 3 **Date:**

DATA VISUALIZATION AND MODELLING USING CLASSIFICATION

Objectives:

- **To visualize the dataset.**
- **To model the data set.**

Data visualization:

RapidMiner provides several plotters in order to visualize the properties of a data set. Each time an ExampleSet is part of the (intermediate) result, the user can select between data view, statistics view and charts view. The charts view provides several 2D and 3D charts, histogram charts, and other charts for high-dimensional data sets.

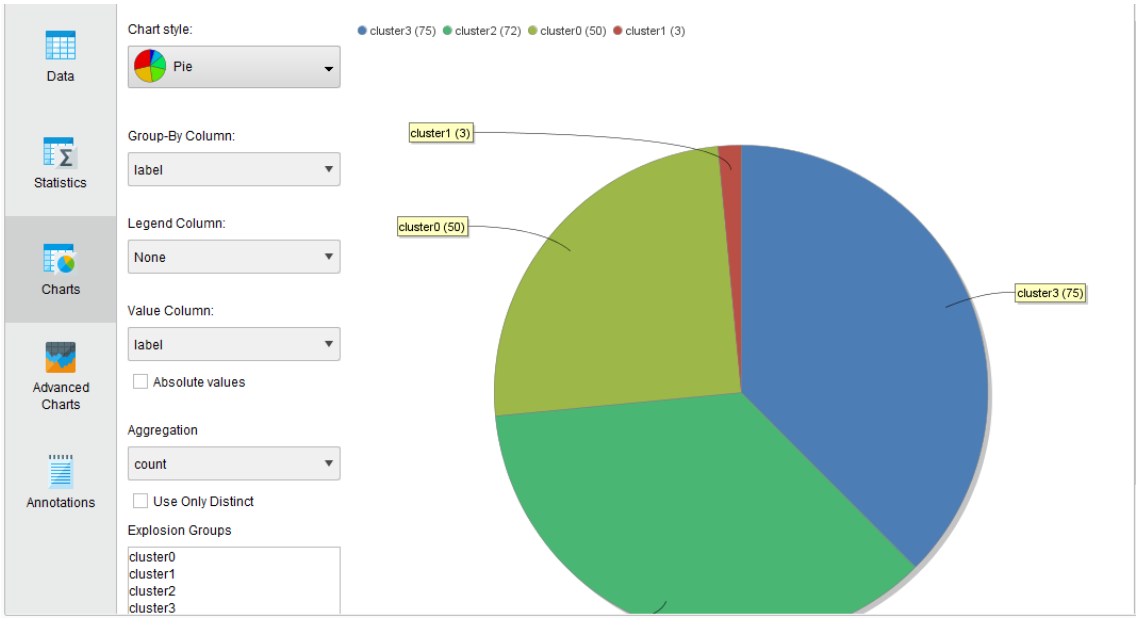


Figure 1. Pie chart for an example data set.

2. **Modelling using classification**
   Decision Tree Model creation

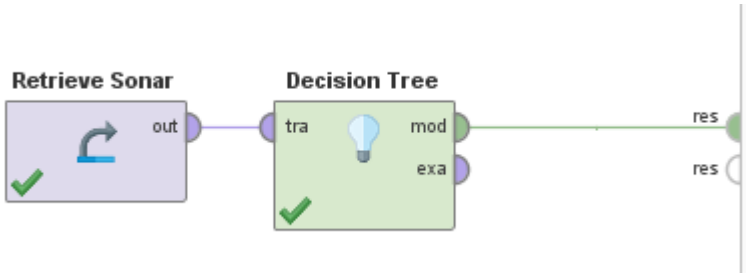


Figure 2. Decision tree model creation

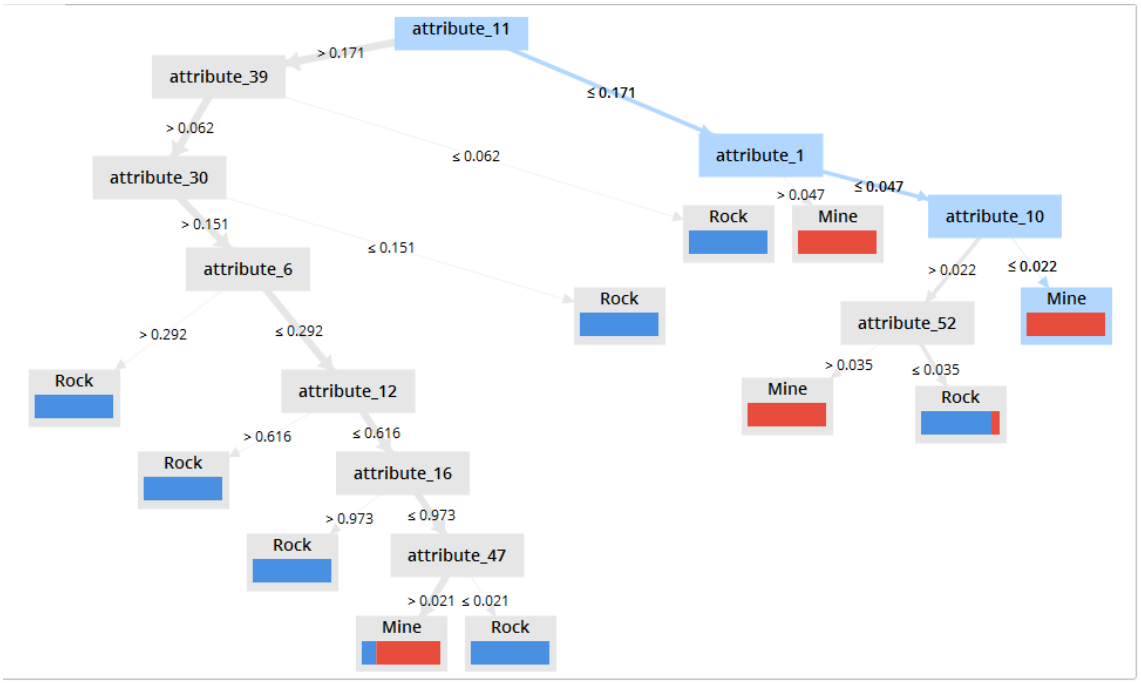


Figure 3. Decision Tree

Criterion: Selects the criterion on which Attributes will be selected for splitting.

Pruning: Prunning the decision tree is optional which if selected confidence for pessimistic error calculation of pruning must be provided.

minimal Maximal depth: gain (optional)

The gain of a node is calculated before splitting it. The node is split if its gain is greater than the minimal gain.

minimal leaf size (optional)

The size of a leaf is the number of Examples in its subset. The tree is generated in such a way that every leaf has at least the *minimal leaf size* number of Examples.

minimal size for split (optional)

The size of a node is the number of Examples in its subset. Only those nodes are split whose size is greater than or equal to the *minimal size for split* parameter.

Performance of the created model by applying on training data

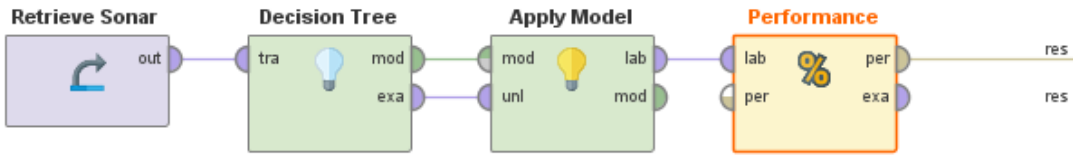


Figure 4. Applying model to traing data

accuracy: 86.06%

	true Rock	true Mine	class precision
pred. Rock	75	7	91.46%
pred. Mine	22	104	82.54%
class recall	77.32%	93.69%	

Figure 5. Accuracy of Training data

Performance on Test data

The Sonar dataset is split into two (80% for training ans 20% for testing) subsets. Since the decision tree model has to be applied on training and test dataset, apply model operator is used on which Performace operator is applied.

In the results, you get two tabs, one for Performance on test data and other for Performance for Training Data.

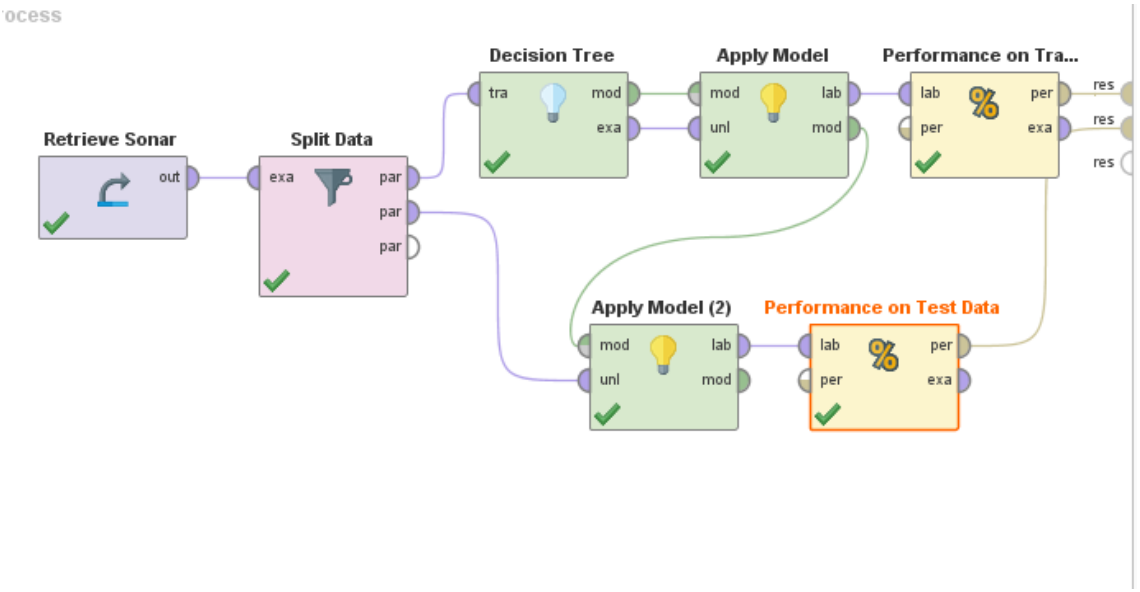


Figure 6. Performance on test data

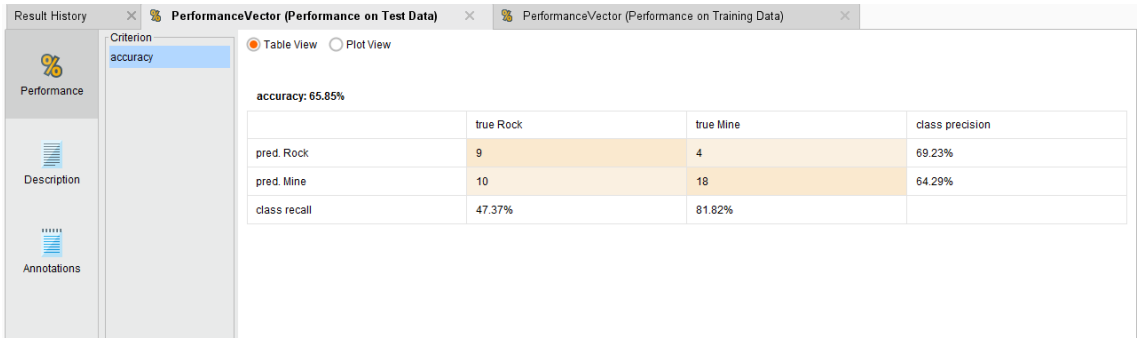


Figure 7. Accuracy on test data

Validation of the Result: Performace is obtained by applying cross validation operator also.

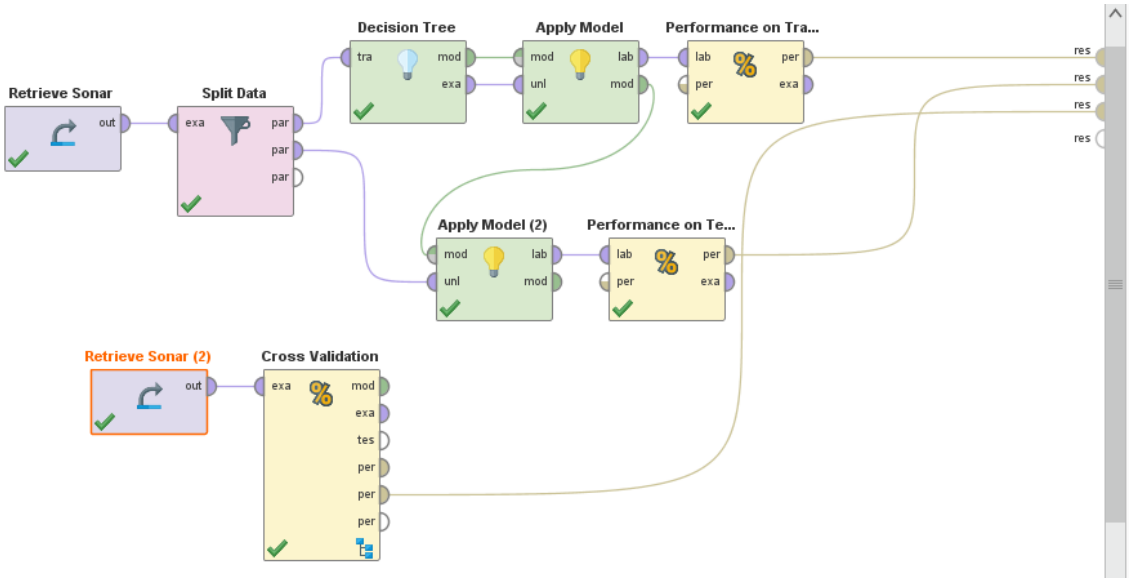


Figure 8. Cross Validation operator

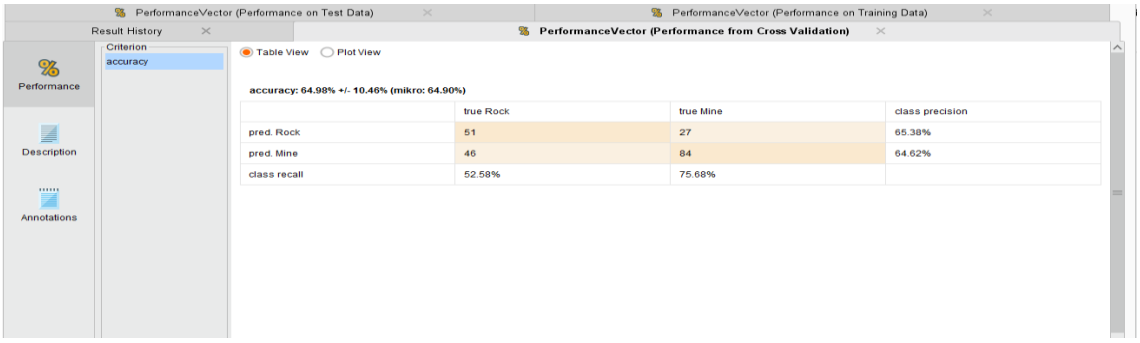


Figure 9. Result after cross validation

Prediction

The example for which the class has be predicted should be given in csv file.

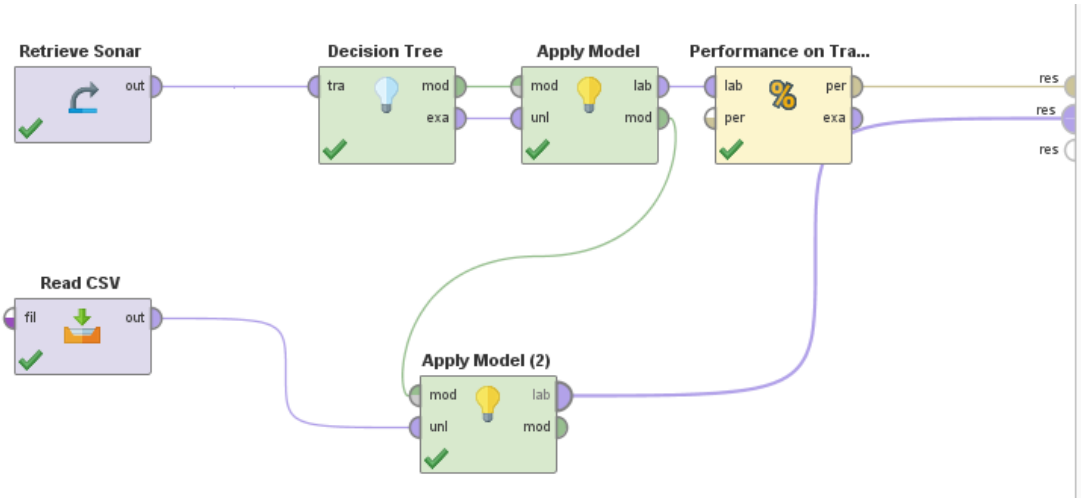


Figure 10. Prediction

Lab exercise:

1. Visualize the Iris dataset with various types of graphs available in rapid miner and say for which configuration of axes, the best class separation is shown.
2. Consider the housing data set with attributes housesize (in sq feet), house price . Find the house price for a house of of 3000 sq. feet using linear regression.
3. Use ID3 operator and build and validate the model.

Additional exercise

Understand and use Naïve Bayes Algorithm. Use this operator for the classification. Validate and predict the result as outlined above in manual.

LAB NO: 4 **Date:**

ASSOCIATION RULE MINING AND CLUSTERING

Objectives:

- **To explore association mining and clustering operators in rapid miner**

Association Mining

FP-Growth:

This operator efciently calculates all frequent itemsets from the given ExampleSet using the FP-tree data structure. It is compulsory that all attributes of the input ExampleSet should be binominal. In simple words, frequent itemsets are groups of items that often appear together in the data. It is important to know the basics of market-basket analysis
for understanding frequent itemsets.

This operator has two basic working modes:

- finding at least the specified number of itemsets with highest support without taking the 'min support' into account. This mode is available when the find min number of itemsets parameter is set to true. Then this operator finds the number of itemsets specified in the min number of itemsets parameter. The min support parameter is ignored in this case.
- finding all itemsets with a support larger than the specified minimum support. The minimum support is specified through the min support parameter. This mode is available when the find min number of itemsets parameter is set to false.

Create Association Rules

This operator generates a set of association rules from the given set of frequent itemsets. Association rules are created by analyzing data for frequent if/then patterns and using the criteria support and confidence to identify the most important relationships. Support is an indication of how frequently the items appear in the database. Confidence indicates the number of times the if/then statements have been found to be true. The frequent if/then

patterns are mined using the operators like the FP-Growth operator. The Create Association Rules operator takes these frequent itemsets and generates association rules.

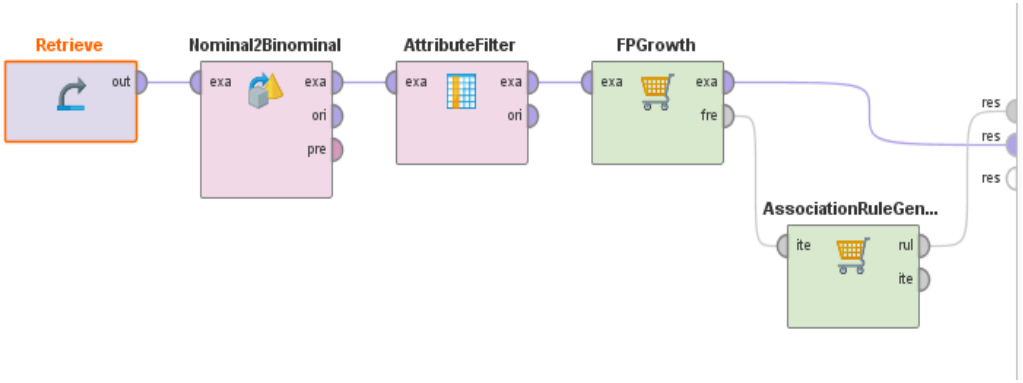


Figure 1. Association rule generation

Apply Association rules:

This operator creates a new confidence attribute for each item occurring in at least one conclusion of an association rule. Then it checks for each example and for each rule, if the example fulfills the premise of the rule, which it does, if it covers all items in the premise. An example covers an item, if the attribute representing the item contains the positive value. If the check is positive, a confidence value for each item in the conclusion is derived. Which value is used, depends on the selected confidence aggregation method. There are two types: The binary choice will set a 1, for any item contained inside a fulfilled rule's conclusion. This is independent of how confident the rule was. Any aggregation choice will select the maximum of the previous and the new value of the selected confidence function.

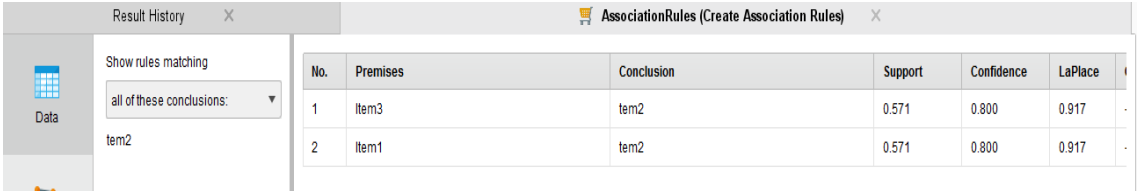


Figure 2. Frequent items in data tab

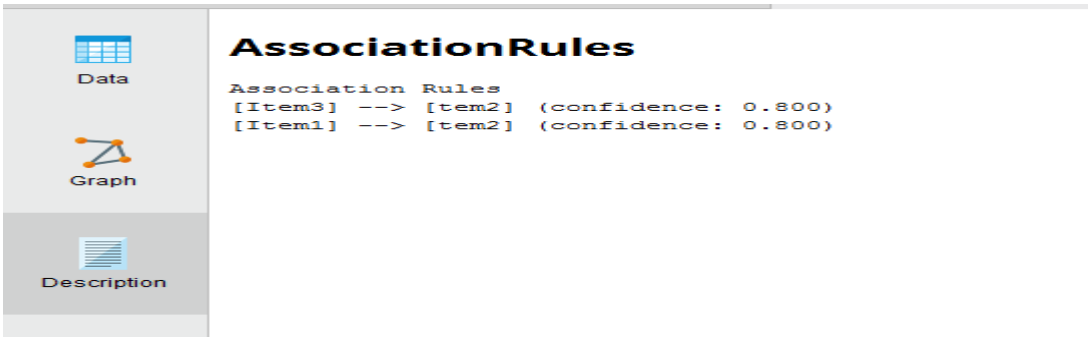


Figure 3. Association Rules in Description tab

Clustering

K-Means: This operator performs clustering using the k-means algorithm. Clustering is concerned with grouping objects together that are similar to each other and dissimilar to the objects belonging to other clusters. Clustering is a technique for extracting information from unlabelled data. k-means clustering is an exclusive clustering algorithm i.e. each object is assigned to precisely one of a set of clusters.

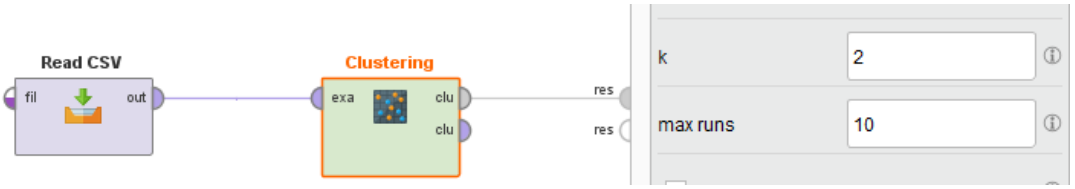


Figure 4. K-Means operator applied on example data set

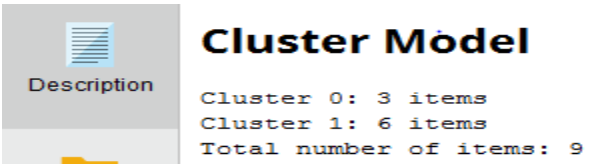


Figure 5. Resulting cluster details

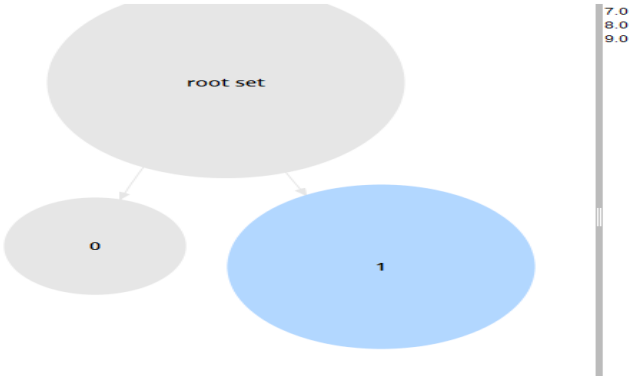


Figure 6. Clusters visualized on click of each circle data points are shown

LAB NO: 5 **Date:**

CREATING PHYSICAL DATA MODEL WITH IBM-INFOSPHERE

Objective

1. **To get acquainted with Infosphere.**
2. **To create physical data model.**

Introduction

InfoSphere™ Warehouse provides a sample database that contains sales information for a fictional company, and a sample Cubing Services cube that contains sample sales and marketing metadata. Additionally InfoSphere Warehouse provides tutorials that work with the sample database to show how to use the SQL Warehousing, Cubing Services, and Mining features of the product.

InfoSphere™ Warehouse is a suite of products that combines the strength of DB2® Enterprise Edition with a data warehousing infrastructure from IBM®.

You can use InfoSphere Warehouse to build a complete data warehousing solution that includes a highly scalable relational database, data access capabilities, and front-end analysis tools.

The following products are provided in InfoSphere Warehouse:

InfoSphere Warehouse data server

- DB2 Enterprise Server Edition
  - Intelligent Miner™
  - DB2 Query Patroller
  - InfoSphere Federation Server Relational Wrappers

InfoSphere Warehouse application server

- Administration Console and Workload Manager
  - SQL Warehousing (SQW) administration
  - Cubing Services administration
  - Intelligent Mining administration
  - Workload Manager
  - Unstructured Text Analysis

- IBM Data Server Client
- WebSphere® Application Server
- Cubing Services cube server
- Mining Blox®

InfoSphere Warehouse client

- Design Studio
  - SQL Warehousing (SQW) Tool
  - Cubing Services modeling
  - Intelligent Mining tools
  - Unstructured Text Analysis tools
  - Mining Blox tools
- IBM Data Server client
  - DB2 Query Patroller Center
- Intelligent Miner Visualization
- Cubing Services client
- Administration Console Command Line Client

Components of InfoSphere Warehouse

The components of InfoSphere™ Warehouse provide an integrated platform for warehouse administration and for the development of warehouse-based analytics.

The key components of InfoSphere Warehouse are described as follows.

InfoSphere Warehouse Design Studio

Design Studio, which includes the SQL Warehousing Tool, provides a common design environment for creating physical data models, SQL data flows and control flows, and Blox® Builder analytic applications. Design Studio is built on the Eclipse Workbench, which is a development environment that you can customize.

SQL Warehousing Tool

SQL Warehousing Tool is a graphical tool that generates SQL for warehouse maintenance and administration. The SQL Warehousing Tool automatically generates SQL that is based on visual operator flows that

you model in Design Studio. The library of SQL operators covers the in-database data operations that are typically needed to move data between database tables and to populate analytical structures, such as multidimensional cubes. SQL Warehousing Tool complements and works with ETL products from IBM® and other vendors.

InfoSphere Warehouse Administration Console

The Warehouse Administration Console is a Web application from which you can deploy and manage applications, control flows, database resources, and system resources. You can do the following types of tasks in the Administration Console:

Common configuration

Create and manage database and system resources, including driver definitions, log files, and notifications.

SQL warehousing

Run and monitor data warehousing applications and view deployment histories and execution statistics.

WebSphere® Application Server

WebSphere Application Server is a Java-based Web application server that is required by the Administration Console. WebSphere Application Server provides a rich application deployment environment with a complete set of application services, including capabilities for transaction management, security, clustering, performance, availability, connectivity, and scalability.

InfoSphere Warehouse

Design Studio

The InfoSphere Warehouse Design Studio (also referred to as Design Studio, or Design Studio) provides a platform and a set of integrated tools for developing Business Intelligence (BI) solutions. You can use these tools to build, populate, and maintain tables and other structures for data mining and Online Analytical Processing (OLAP) analysis in the context of a DB2 data warehouse.

Design Studio includes the following tools and features:
Integrated physical data modeling, based on InfoSphere Data Architect
SQL Warehousing Tool for data flow and control flow design
Data mining, exploration, and visualization tools
Tools for designing OLAP metadata, MQTs, and cube models
Tools for developing AlphaBlox analytical applications
Integration points with InfoSphere DataStage ETL systems
By integrating these tools, Design Studio offers a fast time-to-value and managed cost for warehouse-based analytics. In this chapter we cover the following topics:

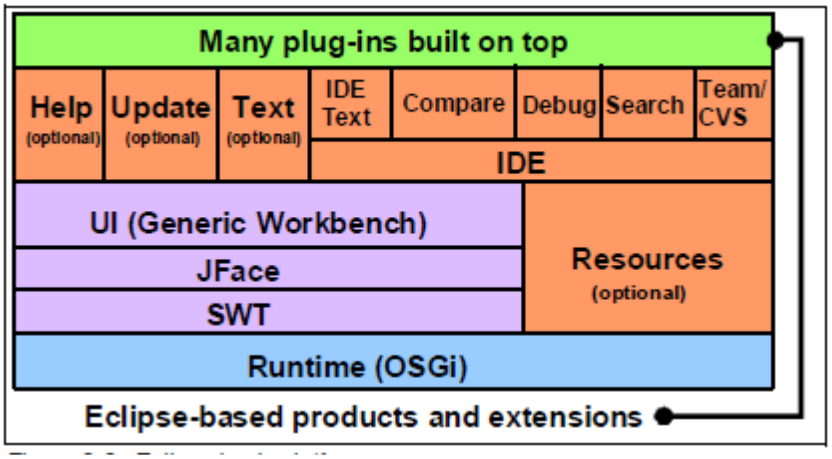
- ✓ The Eclipse Platform
- ✓ The Design Studio Workbench
- ✓ Exploring data
- ✓ Designing physical database models
- ✓ Designing OLAP objects
- ✓ Designing and deploying SQL Warehousing data and control flows

Design Studio supports the BI solutions development life cycle.

The Eclipse platform

Eclipse is an open source community of companies that focus on providing a universal framework for tools integration. The Eclipse consortium was founded in late 2001, and has grown to over 80 members, including many industry-leading software vendors. The Eclipse platform provides a powerful framework and the common graphical user interface (GUI) and infrastructure required to integrate multiple tools easily. The platform is extended by installing plug-ins to provide specific features. InfoSphere Warehouse Design Studio is based upon the open source Eclipse platform.

The basic architecture of Eclipse provides numerous services that tool developers would have to write if they did not use the Eclipse platform. Eclipse has a rich infrastructure, including components such as a runtime environment, a generic user interface, and a help system.



Tool vendors that use Eclipse are able to develop their products quickly. It enables them to focus on their core competency because they only need to build the features in their specialty. The additional capabilities that the tool vendors provide are delivered as plug-ins, which are installed into an existing Eclipse environment. Each of the capabilities that Design Studio provides are packaged together and are installed as plug-ins on top of the basic Eclipse platform.

Users of Eclipse-based tools enjoy many benefits:

_ A rich user experience that is common across all Eclipse-based products (such as InfoSphere Warehouse Design Studio, WebSphere development tools, and the suite of Rational tools).

_ A wide array of instructional resources on the Internet that explain how to extend the Eclipse platform or write tools for it.

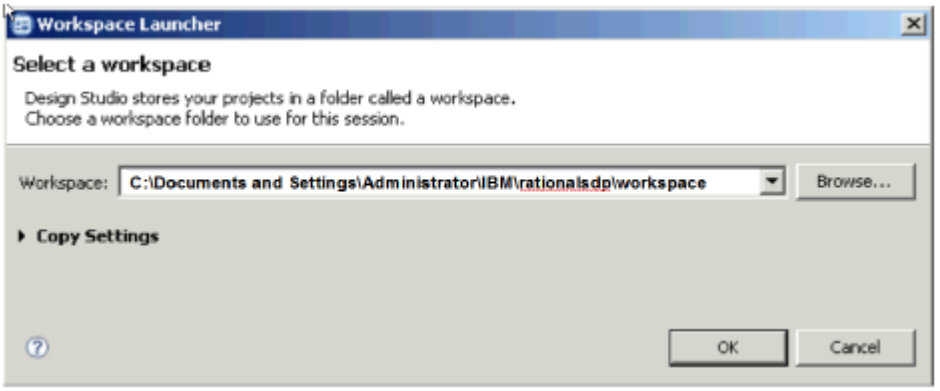
_ A broad selection of third-party tools that have been developed and are available to be installed into InfoSphere Warehouse Design Studio.

Introduction to Design Studio

In this section we provide an introduction to InfoSphere Warehouse Design Studio. The Workbench is the interface you use to access and use the capabilities of Design Studio. But before you can begin using the Workbench, there are basic concepts with which you must become familiar.

The workspace

Every time InfoSphere Warehouse Design Studio is launched, you are prompted to provide a path to the workspace, as shown in Figure below. A workspace is a collection of resources and is the central repository for your data files.



InfoSphere Warehouse Design Studio, like other Eclipse-based tools, helps manage various resources. These resources take the form of projects, folders, and files.

A project is a container used to organize resources pertaining to a specific subject area. The workspace resources are displayed in a tree structure with projects containing folders and files being at the highest level. However, projects do not contain other projects. You can specify different workspaces for different projects, but only one workspace is active per running instance of Design Studio. To change workspaces, choose File->Switch Workspace. A workspace can hold multiple projects.

If you specify a local directory for your workspace, back up that directory on a regular basis.

Projects and the local file system

When you create a new project, you find a new subdirectory on disk, located under the workspace directory specified at start up. In the project directory is a special file with a .project extension. The .project file holds metadata about the project, including information that can be viewed by selecting the Properties view in Design Studio. Inside the project subdirectory you see all of the files and folders that have been created as part of the project. The file names and content are the same, whether accessed from the file system or through Design Studio. You also see a folder with a .metadata extension, located in the workspace directory, at the same level

as the projects that are part of that workspace. The .metadata directory holds platform-specific information, (including workspace structure information). The contents of this directory should never be altered or manipulated outside of the Design Studio API. The project type controls or determines the kinds of objects that are available. In the InfoSphere Warehouse Design Studio, the data design project (OLAP) type allows you to work with physical data models and OLAP objects. The data warehousing project provides entries such as SQL warehousing objects, including data flows, control flows, and mining flows.

The Welcome page

The first time that Design Studio is started in a new workspace, the Welcome page is displayed. A partial view of that page is depicted in Figure below.

The Welcome page contains links to general information, the help system, recorded demos, and tutorials about InfoSphere Warehouse and Design Studio.

InfoSphere Warehouse also provides sample projects that you can work through to become familiar with the workbench, and the steps involved in creating data models, data and control flows, and data mining flows.

Click the Workbench link (the arrow icon displayed at the top of the window) or close the Welcome page to launch the workbench. After you have launched the workbench, you can display the Welcome page by selecting **Help** ->**Welcome**.

The Design Studio Workbench

The term *workbench* refers to the desktop development environment, and delivers the mechanism for navigating the functions provided by the various Design Studio plug-ins. The workbench offers one or more windows, which contain one or more perspectives, views, and editors that allow you to manipulate resources in your project. The default workbench for InfoSphere Warehouse Design Studio contains the following elements:

_ Opened Perspective
_ Menu Bar
_ Project Explorer
_ Editors
_ Palette for Editors
_ Stacked Views
_ State Bar
_ Data Sources Explorer
_ Properties View

Connecting With Data base

In the data source explorer, right click on Database Connections. Select new and connection Parameters such as host name,Database name,username,passwordand database manager as DB2 for linux,unix and windows as per Lab instructor's instruction.

Creating a new DataWarehouse Project

Choose File->new->DatawarehousingProject->give name say "ms1"->finish.

Developing the physical data model

Physical data model

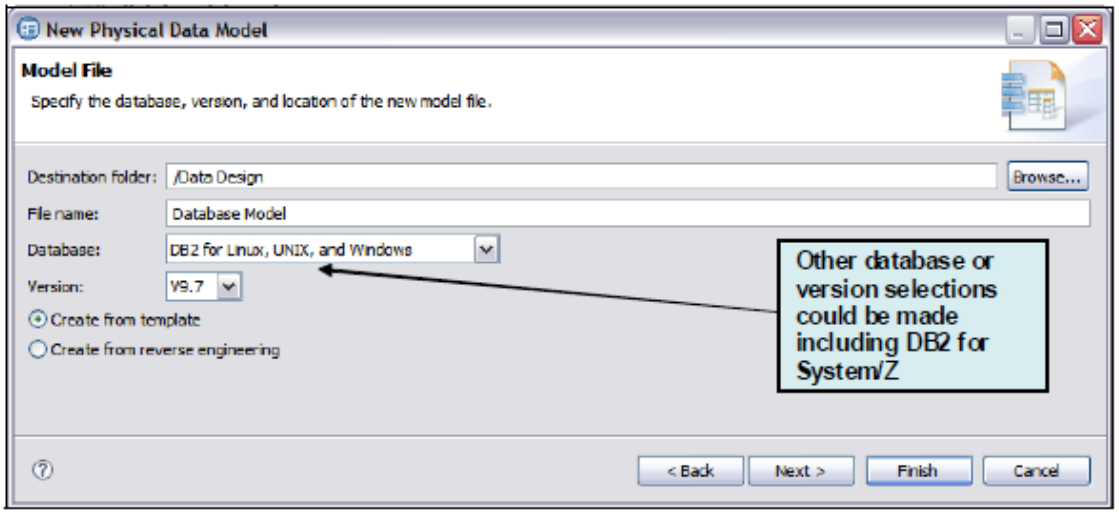
This is the physical implementation of the logical model in a specific database management system. It identifies the implementation details in terms of such things as the configuration, keys, indexes selected, and constraints, and the means by which it is physically stored in the selected database structures.

Physical data models define the internal schema of the data sources in your warehouse environment. Data models outline data tables, the columns in those tables, and the relationships between tables. Design Studio includes the components needed to create a physical model and generate the SQL appropriate for your implementation target. Physical models are constrained to the concept of the target. For example, InfoSphere Warehouse physical models are constrained to the relational model. You can only

model objects that are supported by the target database. For example, the ability to model MQTs only applies to DB2 targets.
The physical models that you create in Design Studio can be used to create brand new schemas or to update schemas that already exist in the target database.
Using Design Studio, in your physical data models, you can define the following data elements for the target database:
Databases (one per data model)
Tables
Views
Primary keys and foreign keys
Indexes
Stored procedures and functions
DB2 Tablespaces and buffer pools
The physical data models that you create and update in Design Studio are implemented as entity relationship diagrams. The models are visually represented using either information Engineering (IE) notation, or Unified Modeling Language (UML) notation. Before you begin your modeling work, configure Design Studio to use the notation that you prefer. Configure Design Studio by clicking Data->Diagram, which takes you to the configuration screen where you can select the desired notation.

Creating Physical Data Model
In the Data Project Explorer, expand “ms1” project you created earlier. You can see several sub folders and icons. In that right click on Data Models option->new->physical data model.You get the following window with ms1 in place of /Data design. Select

Version 9.7 then click **next->Finish**



Now right click on Schema in Data Project Explorer and choose refactor->rename to rename the Schema to a new name to avoid name conflict.

Now right click on new name of Schema and select Add Data Object ->Table->name the table.

Right click on Table and choose Add Data Object->Column and then edit properties of the column of table. Like this you can create tables as many as you require.

Now right click on Diagram->new OverviewDiagram->select Schema check box. You get all tables in Diagram1 tab of editing area.

Now rightclick on Schema in data project explorer, then Analyzemodel->check Physical Data model->apply->Finish

Again rightclick on Schema in Data Project Explorer->Generate DDL->Select All->Deselect Drop statements and UseDomain if exists->next->DeselectAll->Select PrimaryKeyConstraint ,ForeignKey Constraint, Schemas and Tables->next->RunDDL on server->next->Select Database Connection you logged into->next->finish

Adding Data to Tables

In data Source Explorer double click on Database name say GSDB--. Go to Schemas folder expand and then you can see the Schema you created in previous generate DDL command. If you expand it you can see tables folder inside that you can find the tables you have created using Data Project Explorer earlier.

Now you can add data to the fields of each table you created as follows.
Right click on table name->data->edit. Close and save.

Lab exercise

Create Physical model for the following databases by using Infosphere

1. Product(pid,pname)
   Supplier(sid,sname)
   Supply(pid,sid,qty)
2. Student (Regno, Name, Major, Bdate)
   Course (Course_ID, Cname, dept)
   Enroll (RegNo, Course_ID, marks)
   Book_Adoption (Course_ID, sem, ISBN)
   Text (book_ISBN, title, publisher, author)
3. Person (driver _ id , name, address)
   Car (Regno, model, year)
   Accident (report_number, date, location)
   Owns (driver-id, Regno)
   Participated (driver-id, Regno , report_number, damage)
4. Employee (Name, SSN, Salary, DoJoin)
   Project (Pname, Pnumber, Mgr_ssn, PAddress)
   Project_Domain(Dnumber, Pno)
   Domain(Dnumber, Dname, Description)
   Works_On (Essn, Pno, hrs)

LAB NO. 6 **Date**

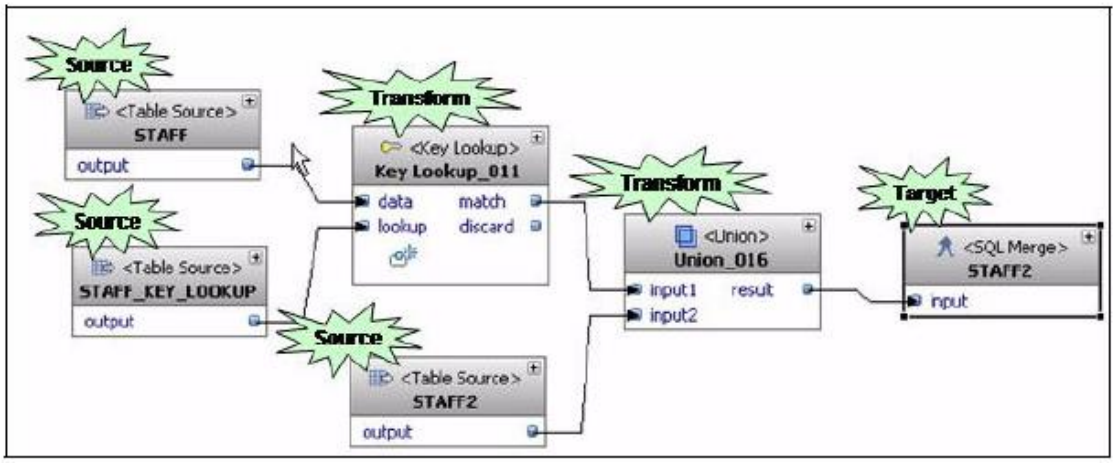
CREATING DATA FLOWS WITH IBM INFOSPHERE

Objectives

1.To learn creating and executing data and control flows

Introduction

Data flows model the SQL-based data movement and transformation activities that are executed by the DB2 database engine. A data flow consists of activities represented by graphical operators that extract data from flat files or relational tables, transform the data, and load it into a relational table in a data warehouse, data mart, or staging area. Data can also be exported to flat files. A sample data flow is shown in Figure below



A control flow sequences and manages the flow of activities. Activities are independent units-of-work, and could be, as examples, a data flow, a database utility, an operating system script, or a stored procedure. A control flow is the unit of execution in the runtime environment.

Steps for creating Data Flow

1. Create the physical model for supplier data base in previous lab exercise.
2. In data project explorer right click on DataFlows->new->DataFlow->select projectname->give a name to data flow ->select Work against

database (online)->next->connection name->finish.

3. Let us create a Dataflow to query the data base for pid and qty.
4. So create a Table by right clicking on schema name and following Add data Object->table-> give name as Result
5. Create columns as required by the Result of query.
6. Generate DDL and see in data source explorer that this table is created.
7. Now from the Palette on R.H.S., expand SQL warehousing operators.
8. Drag a table source to the diagram editor editing area, a window will popup, in that Source data base table as product by selecting from …on R.H.S., of text box.
9. Click next->next->Finish.
10. Similarly drag another table source and link it to supply table.
11. To solve the query we need join operator so select from palette's transformation tab Table join operator.
12. Connect two table sources to input of join in diagram using an arrow which comes automatically when you drag the mouse to connect the blocks.
13. Now doubleclick on join block ->next->click on … ellipse button on right of text box for join condition.
14. Select join condition in our case it is IN_03_0.pid=In1_03_1.pid press ok
15. Now drag Table target and link it to Result table we constructed earlier.
16. You need to select the columns from join as mapping property for the target table icon.
17. Now in Data Project Explorer right click on DataFlow you have constructed and select Validate. If no errors select again execute option in the same menu.
18. Now if you right click in Data source explorer the table Result and choose sample contents you get pid and qty populated from other tables.

Lab exercises

1. Create a data flow by using infosphere to solve the following queries on respective data base created in Lab5.

Student (Regno, Name, Major, Bdate)
Course (Course_ID, Cname, dept)
Enroll (RegNo, Course_ID, marks)
Book_Adoption (Course_ID, sem, ISBN)

Text (book_ISBN, title, publisher, author)

1. Find names and registration numbers of students who were born in the year 1990.
2. Display names of students in alphabetical order who have enrolled in courses of more than 1 department.
3. Find the title of the text books written by author with middle name 'Ken'
4. Display names of the students who have registered for courses for which more than 10 students have registered.

PERSON (driver _ id , name, address)
CAR (Regno, model, year)
ACCIDENT (report_number, date, location)
OWNS (driver-id, Regno)
PARTICIPATED (driver-id, Regno , report_number, damage)

1. Select registration number of cars which do not come in between the model year 2000 and 2010.
2. Find driver names whose sum of damage amount of all his accidents is less than average damage amount of all accidents in the database.
3. Find driver names whose sum of damage amount of all his accidents is less than average damage amount of all accidents in database.
4. Find driver ids' who have met with accident more than once but with different cars owned by him and damage amount is greater than Rs.50000.
5. Find driver id of persons with name 'john'.

Employee (Name, SSN, Salary, DoJoin)
Project (Pname, Pnumber, Mgr_ssn, PAddress)
Project_Domain(Dnumber, Pno)
Domain(Dnumber, Dname, Description)
Works_On (Essn, Pno, hrs)

I. Find names of employees who have joined department between 2010 and 2015.
II. Find the domain with more than 3 projects under it.
III. Find employee names who work on all projects running under domain of "Banking".

Consider the relation Project(EmpNo, Project_No, Hrs, Dept_Name)
Write SQL queries for the following:

1. Find employees who are working in projects numbers P1 and P2.
2. Display each employee number, their working hours along with average working hours of all employees in their department.
3. Delete all employees who are working less than 4 hrs.

Product (P_code, Description, Stocking_date, Qty, Price, Discount, V_code)
Vendor(V_code, Name, Address, Phone)
Here a vendor can supply more than one product but a product is supplied by only one vendor. Write SQL queries for the following:

1. List the names of all the vendors who supply more than one product.
2. List the details of the products whose prices exceed the average product price of all products.

List the Name, Address and Phone of the vendors who are currently not supplying any product.

LAB NO. 7 **Date:**

IMPLEMENTATION OF APRIORI ALGORITHM

Objectives

1. **Explain association mining.**
2. **Generate frequent patterns for association rule mining**
3. **Explain apriori property.**

Introduction

When we go for grocery shopping, we often have a standard list of things to buy. Each shopper has a distinctive list, depending on one's needs and preferences. A housewife might buy healthy ingredients for a family dinner, while a bachelor might buy beer and chips. Understanding these buying patterns can help to increase sales in several ways. If there is a pair of items, X and Y, that are frequently bought together:

- Both X and Y can be placed on the same shelf, so that buyers of one item would be prompted to buy the other.
- Promotional discounts could be applied to just one out of the two items.
- Advertisements on X could be targeted at buyers who purchase Y.
- X and Y could be combined into a new product, such as having Y in flavors of X.

While we may know that certain items are frequently bought together, the question is, how do we uncover these associations?

Besides increasing sales profits, association rules can also be used in other fields. In medical diagnosis for instance, understanding which symptoms tend to co-morbid can help to improve patient care and medicine prescription.

Association rules analysis is a technique to uncover how items are associated to each other.

There are two main measures for measuring association. They are

19. **Support**. This says how popular an itemset is, as measured by the proportion of transactions in which an itemset appears. In Table 1 below, the support of {apple} is 4 out of 8, or 50%. Itemsets can also contain multiple items. For instance, the support of {apple, beer, rice} is 2 out of 8, or 25%.

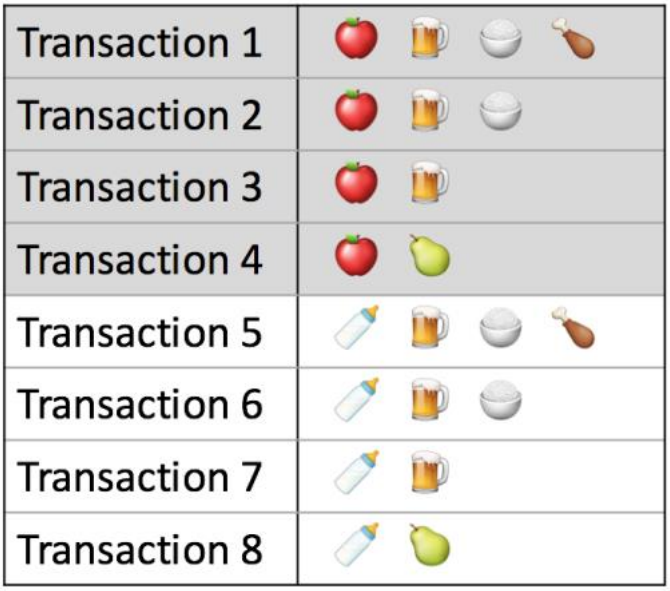


Figure 1.1. Example Transaction Table

If you discover that sales of items beyond a certain proportion tend to have a significant impact on your profits, you might consider using that proportion as your *support threshold*. You may then identify itemsets with support values above this threshold as significant itemsets or frequent itemsets.

$$\text{Support } \{🍎\} = \frac{4}{8}$$

20. **Confidence**. This says how likely item Y is purchased when item X is purchased, expressed as {X -> Y}. This is measured by the proportion of transactions with item X, in which item Y also appears. In Table 1, the confidence of {apple -> beer} is 3 out of 4, or 75%.

$$\text{Confidence } \{🍎 \rightarrow 🍺\} = \frac{\text{Support } \{🍎, 🍺\}}{\text{Support } \{🍎\}}$$

Apriori Principle

The *apriori principle* can reduce the number of itemsets we need to examine. Put simply, the apriori principle states that if an itemset is infrequent, then all its subsets must also be infrequent. This means that if {beer} was found to be infrequent, we can expect {beer,

pizza} to be equally or even more infrequent. So in consolidating the list of popular itemsets, we need not consider {beer, pizza}, nor any other itemset configuration that contains beer.

Apriori algorithm Basics

Key Concepts :

• Frequent Itemsets: The sets of item which has minimum support (denoted by Li for ith-Itemset).

• Apriori Property: Any subset of frequent itemset must be frequent.

• Join Operation: To find L_k , a set of candidate k-itemsets is generated by joining L_{k-1} with itself.

Apriori Algorithm Pseudocode

```
procedure Apriori (T, minSupport) { //T is the database and minSupport is the minimum support
        L1= {frequent items};
        for (k= 2; Lk-1 !=∅; k++) {
                Ck= candidates generated from Lk-1
                //that iscartesian product Lk-1 x Lk-1 and eliminating any k-1 size itemset that is not
                //frequent
                for each transaction t in database do{
                        #increment the count of all candidates in Ck that are contained in t
                        Lk = candidates in Ck with minSupport
                }//end for each
        }//end for
        return ∪k Lk;
}
```

Example

A large supermarket tracks sales data by Stock-keeping unit (SKU) for each item, and thus is able to know what items are typically purchased together. Apriori is a moderately efficient way to build a list of frequent purchased item pairs from this data. Let the database of transactions consist of the sets {1,2,3,4},{1,2,3,4,5}, {2,3,4}, {2,3,5}, {1,2,4}, {1,3,4}, {2,3,4,5}, {1,3,4,5}, {3,4,5}, {1,2,3,5}. Each number corresponds to a product such as "butter" or "water". The first step of Apriori is to count up the frequencies, called the supports, of each member item separately:

Item Support
1 6
2 7
3 9
4 8
5 6

We can define a minimum support level to qualify as "frequent," which depends on the context.For this case, let min support = 4. Therefore, all are frequent. The next step is to generate a list of all 2-pairs of the frequent items. Had any of the above items not been frequent, they wouldn't have been included as a possible member of possible 2-item pairs. In this way, Apriori prunes the tree of all possible sets. In next step we again select only these items (now 2-pairs are items) which are frequent (the pairs written in bold text):

Item Support
{1,2} 4
{1,3} 5
{1,4} 5
{1,5} 3
{2,3} 6
{2,4} 5
{2,5} 4
{3,4} 7
{3,5} 6
{4,5} 4

We generate the list of all 3-triples of the frequent items (by connecting frequent pair with frequent single item).

Item Support
{1,3,4} 4
{2,3,4} 4
{2,3,5} 4
{3,4,5} 4

The algorithm will end here because the pair {2,3,4,5} generated at the next step does not have the desired support.

We will now apply the same algorithm on the same set of data considering that the min

support is 5. We get the following results:
Step 1:
Item Support
1 6
2 7
3 9
4 8
5 6
Step 2:
Item Support
{1,2} 4
{1,3} 5
{1,4} 5
{1,5} 3
{2,3} 6
{2,4} 5
{2,5} 4
{3,4} 7
{3,5} 6
{4,5} 4
The algorithm ends here because none of the 3-triples generated at Step 3 have no desired support.

Lab exercises

1. Implement the above Apriori algorithm

Additional exercises

1. Implement Pincer Search algorithm
2. Implement Dynamic Item Set algorithm

LAB NO. 8 **Date:**

IMPLEMENTATION OF APRIORI ALGORITHM

LAB NO: 9 **Date:**

IMPLEMENTATION OF K-MEANS ALGORITHM

Objectives

1. To study what is clustering
2. Implement K-means algorithm for clustering

Introduction

Clustering is the task of partitioning the points into natural groups called clusters, such that points within a group are very similar, whereas points across clusters are as dissimilar as possible. Depending on the data and desired cluster characteristics, there are different types of clustering paradigms such as partitioning, hierarchical, density-based, graph-based, and spectral clustering.

The simplest and most fundamental version of cluster analysis is partitioning, which organizes the objects of a set into several exclusive groups or clusters. To keep the problem specification concise, we can assume that the number of clusters is given as background knowledge. This parameter is the starting point for partitioning methods. Formally, given a data set, D, of n objects, and k, the number of clusters to form, a partitioning algorithm organizes the objects into k partitions .k <=n, where each partition represents a cluster. The clusters are formed to optimize an objective partitioning criterion, such as a dissimilarity function based on distance, so that the objects within a cluster are “similar” to one another and “dissimilar” to objects in other clusters in terms of the data set attributes.

k-Means: A Centroid-Based Technique

Suppose a data set, D, contains n objects in Euclidean space. Partitioning methods distribute the objects in D into k clusters, C1, …, Ck, that is, Ci belongs to D and Ci intersection Cj is empty set, for 1 <=i, j <=k. An objective function is used to assess the partitioning quality so that objects within a cluster are similar to one another but dissimilar to objects in other clusters. This is, the objective function aims for high intracluster similarity and low intercluster similarity.

A centroid-based partitioning technique uses the centroid of a cluster, Ci , to represent that cluster. Conceptually, the centroid of a cluster is its center point. The centroid can be defined in various ways such as by the mean or medoid of the objects (or points)

assigned to the cluster. The difference between an object p belongs to Ci and ci, the representative of the cluster, is measured by dist(p, ci), where dist(x,y) is the Euclidean distance between two points x and y. The quality of cluster Ci can be measured by the within cluster variation, which is the sum of squared error between all objects in Ci and the centroid ci, defined as

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} dist(\boldsymbol{p}, \boldsymbol{c_i})^2,$$

where *E* is the sum of the squared error for all objects in the data set; *p* is the point in space representing a given object; and *ci* is the centroid of cluster *Ci* (both *p* and *ci* are multidimensional). In other words, for each object in each cluster, the distance from the object to its cluster center is squared, and the distances are summed. This objective function tries to make the resulting *k* clusters as compact and as separate as possible. Optimizing the within-cluster variation is computationally challenging. In the worst case, we would have to enumerate a number of possible partitionings that are exponential to the number of clusters, and check the within-cluster variation values. It has been shown that the problemis NP-hard in general Euclidean space even for two clusters (i.e., *k* =2).Moreover, the problem is NP-hard for a general number of clusters *k* even in the 2-D Euclidean space. If the number of clusters *k* and the dimensionality of the space *d* are fixed, the problem can be solved in time $O(n^{dk+1} \log n)$, where *n* is the number of objects. To overcome the prohibitive computational cost for the exact solution, greedy approaches are often used in practice. A prime example is the *k*-means algorithm, which is simple and commonly used.

"How does the k-*means algorithm work?"* The *k*-means algorithm defines the centroid of a cluster as the mean value of the points within the cluster. It proceeds as follows. First, it randomly selects *k* of the objects in *D*, each of which initially represents a cluster mean or center. For each of the remaining objects, an object is assigned to the cluster to which it is the most similar, based on the Euclidean distance between the object and the cluster mean. The *k*-means algorithm then iteratively improves the within-cluster variation. For each cluster, it computes the new mean using the objects assigned to the cluster in the previous iteration. All the objects are then reassigned using the updated means as the new cluster centers. The iterations continue until the assignment is stable, that is,

the clusters formed in the current round are the same as those formed in the previous round.

Algorithm: *k*-means. The *k*-means algorithm for partitioning, where each cluster's center
is represented by the mean value of the objects in the cluster.

Input:

k: the number of clusters,

D: a data set containing *n* objects.

Output: A set of *k* clusters.

Method:

(1) arbitrarily choose *k* objects from *D* as the initial cluster centers;

(2) **repeat**

(3) (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;

(4) update the cluster means, that is, calculate the mean value of the objects for each cluster;

(5) **until** no change;

As a simple illustration of a k-means algorithm, consider the following data set consisting of the scores of two variables on each of seven individuals:

Subject	A	B
1	1.0	1.0
2	1.5	2.0
3	3.0	4.0
4	5.0	7.0
5	3.5	5.0
6	4.5	5.0
7	3.5	4.5

This data set is to be grouped into two clusters. As a first step in finding a sensible initial partition, let the A & B values of the two individuals furthest apart (using the Euclidean distance measure), define the initial cluster means, giving:

	Individual	Mean Vector (centroid)
Group 1	1	(1.0, 1.0)
Group 2	4	(5.0, 7.0)

The remaining individuals are now examined in sequence and allocated to the cluster to which they are closest, in terms of Euclidean distance to the cluster mean. The mean vector is recalculated each time a new member is added. This leads to the following series of steps:

	Cluster 1		Cluster 2	
Step	Individual	Mean Vector (centroid)	Individual	Mean Vector (centroid)
1	1	(1.0, 1.0)	4	(5.0, 7.0)
2	1, 2	(1.2, 1.5)	4	(5.0, 7.0)
3	1, 2, 3	(1.8, 2.3)	4	(5.0, 7.0)
4	1, 2, 3	(1.8, 2.3)	4, 5	(4.2, 6.0)
5	1, 2, 3	(1.8, 2.3)	4, 5, 6	(4.3, 5.7)
6	1, 2, 3	(1.8, 2.3)	4, 5, 6, 7	(4.1, 5.4)

Now the initial partition has changed, and the two clusters at this stage having the following characteristics:

	Individual	Mean Vector (centroid)
Cluster 1	1, 2, 3	(1.8, 2.3)
Cluster 2	4, 5, 6, 7	(4.1, 5.4)

But we cannot yet be sure that each individual has been assigned to the right cluster. So, we compare each individual's distance to its own cluster mean and to that of the opposite cluster. And we find:

	Individual	Mean Vector (centroid)
Cluster 1	1, 2	(1.3, 1.5)
Cluster 2	3, 4, 5, 6, 7	(3.9, 5.1)

The iterative relocation would now continue from this new partition until no more relocations occur. However, in this example each individual is now nearer its own cluster mean than that of the other cluster and the iteration stops, choosing the latest partitioning as the final cluster solution.

Also, it is possible that the k-means algorithm won't find a final solution. In this case it would be a good idea to consider stopping the algorithm after a pre-chosen maximum of iterations.

Lab exercises

1. **Implement K-means algorithm**

Additional exercises

1. **Implement K-medoid algorithm**

LAB NO: 10 **Date:**

IMPLEMENTATION OF DECISION TREE ALGORITHM

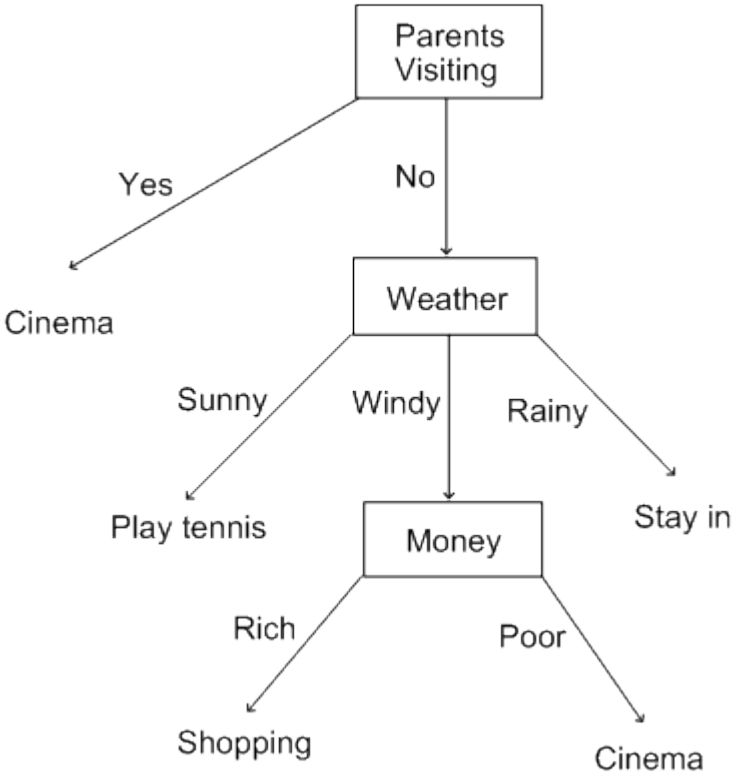
Objectives

Introduction

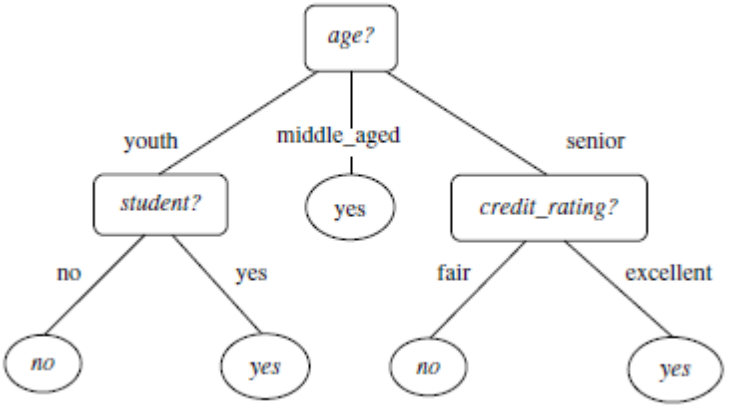
A decision tree is a graphical representation of possible solutions to a decision based on certain conditions. It's called a decision tree because it starts with a single box (or root), which then branches off into a number of solutions, just like a tree.
Decision trees are helpful, not only because they are graphics that help you 'see' what you are thinking, but also because making a decision tree requires a systematic, documented thought process. Often, the biggest limitation of our decision making is that we can only select from the known alternatives. Decision trees help formalize the brainstorming process so we can identify more potential solutions.
You are making your weekend plans and find out that your parents might come to town. You'd like to have plans in place, but there are a few unknown factors that will determine what you can, and can't, do. Time for a decision tree.

First, you draw your decision box. This is the box that includes the event that starts your decision tree. In this case it is your parents coming to town. Out of that box, you have a branch for each possible outcome. In our example, it's easy: yes or no - either your parents come or they don't.

Your parents love the movies, so if they come to town, you'll go to the cinema. Since the goal of the decision tree is to decide your weekend plans, you have an answer. But, what about if your parents don't come to town? We can go back up to the 'no branch' from the decision box and finish our decision tree. You can have other options too elaborated and one such sample decision tree is as follows.



Decision tree induction is the learning of decision trees from class-labeled training tuples. A decision tree is a flowchart-like tree structure, where each internal node (non leaf node) denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (or terminal node) holds a class label. The topmost node in a tree is the root node. A typical decision tree is shown in Figure below. It represents the concept buys computer, that is, it predicts whether a customer at AllElectronics is likely to purchase a computer. Internal nodes are denoted by rectangles, and leaf nodes are denoted by ovals. Some decision tree algorithms produce only *binary* trees (where each internal node branches to exactly two other nodes), whereas others can produce nonbinary trees.



"How are decision trees used for classification?" Given a tuple, ***X***, for which the associated class label is unknown, the attribute values of the tuple are tested against the decision tree. A path is traced from the root to a leaf node, which holds the class prediction for that tuple. Decision trees can easily be converted to classification rules.

"Why are decision tree classifiers so popular?" The construction of decision tree classifiers does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle multidimensional data. Their representation of acquired knowledge in tree form is intuitive and generally easy to assimilate by humans. The learning and classification steps of decision tree induction are simple and fast. In general, decision tree classifiers have good accuracy.

However, successful use may depend on the data at hand. Decision tree induction algorithms have been used for classification in many application areas such as medicine, manufacturing and production, financial analysis, astronomy, and molecular biology. Decision trees are the basis of several commercial rule induction systems.

Algorithm: Generate decision tree. Generate a decision tree from the training tuples of
data partition, *D*.

Input:

Data partition, *D*, which is a set of training tuples and their associated class labels;
attribute list, the set of candidate attributes;
Attribute selection method, a procedure to determine the splitting criterion that "best"

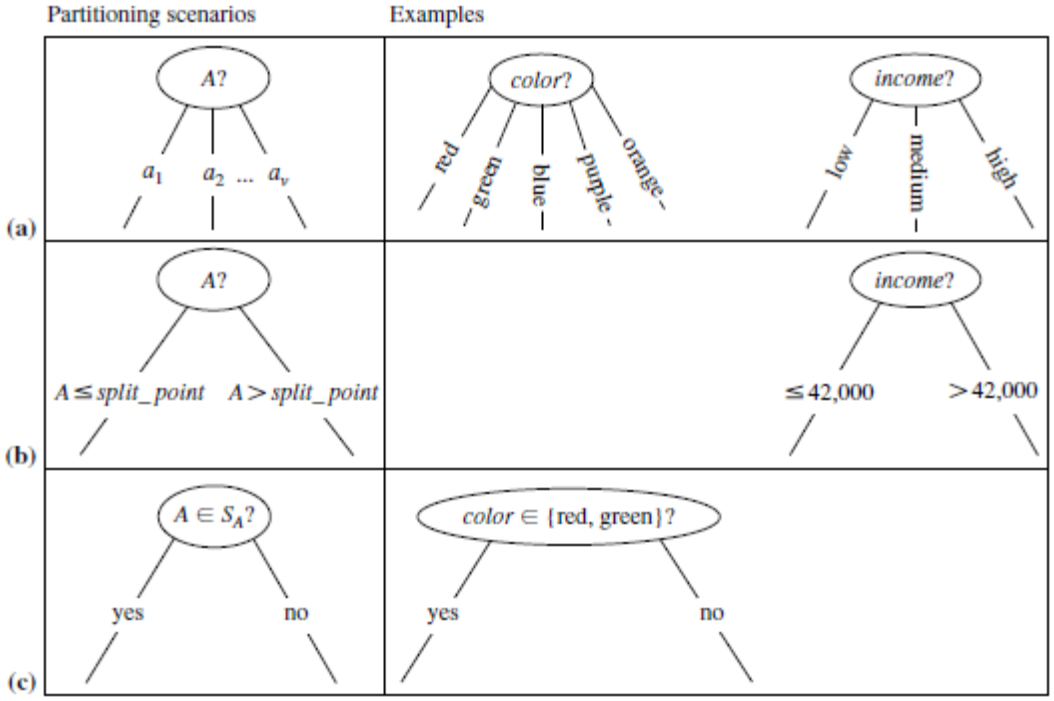
partitions the data tuples into individual classes. This criterion consists of a *splitting attribute* and, possibly, either a *split-point* or *splitting subset*.
Output: A decision tree.
Method:
(1) create a node *N*;
(2) **if** tuples in *D* are all of the same class, *C*, **then**
(3) return *N* as a leaf node labeled with the class *C*;
(4) **if** *attribute list* is empty **then**
(5) return *N* as a leaf node labeled with the majority class in *D*; // majority voting
(6) apply **Attribute selection method**(*D*, *attribute list*) to **find** the "best" *splitting criterion*;
(7) label node *N* with *splitting criterion*;
(8) **if** *splitting attribute* is discrete-valued **and**
multiway splits allowed **then** // not restricted to binary trees
(9) *attribute list attribute list* □ *splitting attribute*; // remove *splitting attribute*
(10) **for each** outcome *j* of *splitting criterion*
// partition the tuples and grow subtrees for each partition
(11) let *Dj* be the set of data tuples in *D* satisfying outcome *j*; // a partition
(12) **if** *Dj* is empty **then**
(13) attach a leaf labeled with the majority class in *D* to node *N*;
(14) **else** attach the node returned by **Generate decision tree**(*Dj* , *attribute list*) to node *N*;
endfor
(15) return *N*;

The algorithm is called with three parameters: *D*, *attribute list*, and *Attribute selection method*. We refer to *D* as a data partition. Initially, it is the complete set of training tuples and their associated class labels. The parameter *attribute list* is a list of attributes describing the tuples. *Attribute selection method* specifies a heuristic procedure for selecting the attribute that "best" discriminates the given tuples according to class. This procedure employs an attribute selection measure such as information gain or the Gini index. Whether the tree is strictly binary is generally driven by the attribute selection measure. Some attribute selection measures, such as the Gini index, enforce the resulting tree to be binary. Others, like information gain, do not, therein allowing multiway splits (i.e., two or more branches to be grown from a node).

The tree starts as a single node, *N*, representing the training tuples in *D* (step 1). If the tuples in D are all of the same class, then node N becomes a leaf and is labelled with that

class (steps 2 and 3). Note that steps 4 and 5 are terminating conditions. All terminating conditions are explained at the end of the algorithm. Otherwise, the algorithm calls Attribute selection method to determine the splitting criterion. The splitting criterion tells us which attribute to test at node N by determining the "best" way to separate or partition the tuples in D into individual classes (step 6). The splitting criterion also tells us which branches to grow from node N with respect to the outcomes of the chosen test. More specifically, the splitting criterion indicates the splitting attribute and may also indicate either a split-point or a splitting subset. The splitting criterion is determined so that, ideally, the resulting partitions at each branch are as "pure" as possible. A partition is **pure** if all the tuples in it belong to the same class. In other words, if we split up the tuples in *D* according to the mutually exclusive outcomes of the splitting criterion, we hope for the resulting partitions to be as pure as possible.

The node *N* is labelled with the splitting criterion, which serves as a test at the node (step 7). A branch is grown from node *N* for each of the outcomes of the splitting criterion. The tuples in *D* are partitioned accordingly (steps 10 to 11). There are three possible scenarios,as indicated in figure below.



Let *A* be the splitting attribute. *A* has *v* distinct values, {*a*1, *a*2,…,av} based on the training data.

1. *A is discrete-valued:* In this case, the outcomes of the test at node *N* correspond directly to the known values of *A*. A branch is created for each known value, *aj* , of *A* and labeled with that value. Partition *Dj* is the subset of class-labeled tuples in *D* having value *aj* of *A*. Because all the tuples in a given partition have the same value for *A*, *A* need not be considered in any future partitioning of the tuples. Therefore, it is removed from *attribute list* (steps 8 and 9).

2. *A is continuous-valued:* In this case, the test at node *N* has two possible outcomes, corresponding to the conditions *A* <=*split point* and *A* > *split point*, respectively, where *split point* is the split-point returned by *Attribute selection method* as part of the splitting criterion. (In practice, the split-point, *a*, is often taken as the midpoint of two known adjacent values of *A* and therefore may not actually be a pre-existing value of *A* from the training data.) Two branches are grown from *N* and labelled according to the previous outcomes. The tuples are partitioned such that *D*1 holds the subset of class-labelled tuples in *D* for which *A* <=*split point*, while *D*2 holds the rest.

3. *A is discrete-valued* and a *binary tree* must be produced (as dictated by the attribute selection measure or algorithm being used): The test at node *N* is of the form "*A belongs to SA*?," where *SA* is the splitting subset for *A*, returned by *Attribute selection method* as part of the splitting criterion. It is a subset of the known values of *A*. If a given tuple has value *aj* of *A* and if *aj* belongs to *SA*, then the test at node *N* is satisfied. Two branches are grown from *N* By convention, the left branch out of *N* is labelled *yes* so that *D*1 corresponds to the subset of class-labelled tuples in *D* that satisfy the test. The right branch out of *N* is labelled *no* so that *D*2 corresponds to the subset of class-labelled tuples from *D* that do not satisfy the test.

The algorithm uses the same process recursively to form a decision tree for the tuples at each resulting partition, *Dj* ,of *D* (step 14).

The recursive partitioning stops only when any one of the following terminating conditions is true:

1. All the tuples in partition *D* (represented at node *N*) belong to the same class (steps 2 and 3).

2. There are no remaining attributes on which the tuples may be further partitioned (step 4). In this case, **majority voting** is employed (step 5). This involves converting node *N* into a leaf and labeling it with the most common class in *D*.
Alternatively, the class distribution of the node tuples may be stored.

3. There are no tuples for a given branch, that is, a partition *Dj* is empty (step 12).
In this case, a leaf is created with the majority class in *D* (step 13).

The resulting decision tree is returned (step 15).

Attribute Selection Measures

An attribute selection measure is a heuristic for selecting the splitting criterion that "best" separates a given data partition, *D*, of class-labelled training tuples into individual classes. If we were to split *D* into smaller partitions according to the outcomes of the splitting criterion, ideally each partition would be pure (i.e., all the tuples that fall into a given partition would belong to the same class). Conceptually, the "best" splitting criterion
is the one that most closely results in such a scenario. Attribute selection measures are also known as splitting rules because they determine how the tuples at a given node are to be split.

The attribute selection measure provides a ranking for each attribute describing the given training tuples. The attribute having the best score for the measure4 is chosen as the *splitting attribute* for the given tuples. If the splitting attribute is continuous-valued or if we are restricted to binary trees, then, respectively, either a *split point* or a *splitting subset* must also be determined as part of the splitting criterion. The tree node created for partition *D* is labeled with the splitting criterion, branches are grown for each outcome of the criterion, and the tuples are partitioned accordingly. This section describes three popular attribute selection measures—*information gain, gain ratio,* and *Gini index.*

The notation used herein is as follows. Let *D*, the data partition, be a training set of class-labeled tuples. Suppose the class label attribute has *m* distinct values defining *m* distinct classes, *Ci* (for i =1…, m). Let $C_{i,D}$ be the set of tuples of class *Ci* in *D*. Let |*D*|and $|C_{i,D}|$ denote the number of tuples in *D* and $C_{i,D}$, respectively.

Information Gain

ID3 uses **information gain** as its attribute selection measure. This measure is based on pioneering work by Claude Shannon on information theory, which studied the value or "information content" of messages. Let node *N* represent or hold the tuples of partition *D*. The attribute with the highest information gain is chosen as the splitting attribute for node *N*. This attribute minimizes the information needed to classify the tuples in the resulting partitions and reflects the least randomness or "impurity" in these partitions. Such an approach minimizes the expected number of tests needed to classify a given tuple and guarantees that a simple (but not necessarily the simplest) tree is found.

The expected information needed to classify a tuple in *D* is given by

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i),$$

where *pi* is the nonzero probability that an arbitrary tuple in *D* belongs to class *Ci* and is estimated by $|C_{i,D}|/|D|$. A log function to the base 2 is used, because the information is encoded in bits. *Info*(*D*) is just the average amount of information needed to identify the class label of a tuple in *D*. Note that, at this point, the information we have is based solely on the proportions of tuples of each class. *Info*(*D*) is also known as the **entropy** of *D*. Now, suppose we were to partition the tuples in *D* on some attribute *A* having *v* distinct values, {*a*1, *a*2, : : : , *av*}, as observed from the training data. If *A* is discrete-valued, these values correspond directly to the *v* outcomes of a test on *A*. Attribute *A* can be used to split *D* into *v* partitions or subsets,{ D1, *D*2, … , *Dv*}, where *Dj* contains those tuples in *D* that have outcome *aj* of *A*. These partitions would correspond to the branches grown from node *N*. Ideally, we would like this partitioning to produce an exact classification of the tuples. That is, we would like for each partition to be pure. However, it is quite likely that the partitions will be impure (e.g., where a partition may contain a collection of tuples from different classes rather than from a single class).
How much more information would we still need (after the partitioning) to arrive at an exact classification? This amount is measured by

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j).$$

The term
|*Dj* | / |*D*| acts as the weight of the *j*th partition. *InfoA*.(*D*) is the expected information required to classify a tuple from *D* based on the partitioning by *A*. The smaller the expected information (still) required, the greater the purity of the partitions. Information gain is defined as the difference between the original information requirement (i.e., based on just the proportion of classes) and the new requirement (i.e., obtained after partitioning on *A*). That is,

$$Gain(A) = Info(D) - Info_A(D).$$

In other words, *Gain*(*A*) tells us how much would be gained by branching on *A*. It is the expected reduction in the information requirement caused by knowing the value of

A. The attribute *A* with the highest information gain, *Gain*(*A*), is chosen as the splitting attribute at node *N*. This is equivalent to saying that we want to partition on the attribute *A* that would do the "best classification," so that the amount of information still required to finish classifying the tuples is minimal (i.e., minimum $Info_A(D)$).

Table 8.1 Class-Labeled Training Tuples from the *AllElectronics* Customer Database

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

Example 8.1 **Induction of a decision tree using information gain.** Table 8.1 presents a training set, *D*, of class-labeled tuples randomly selected from the *AllElectronics* customer database. (The data are adapted from Quinlan [Qui86]. In this example, each attribute is discrete-valued. Continuous-valued attributes have been generalized.) The class label attribute, *buys_computer*, has two distinct values (namely, {*yes*, *no*}); therefore, there are two distinct classes (i.e., $m = 2$). Let class C_1 correspond to *yes* and class C_2 correspond to *no*. There are nine tuples of class *yes* and five tuples of class *no*. A (root) node *N* is created for the tuples in *D*. To find the splitting criterion for these tuples, we must compute the information gain of each attribute. We first use Eq. (8.1) to compute the expected information needed to classify a tuple in *D*:

$$Info(D) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940 \text{ bits}.$$

Next, we need to compute the expected information requirement for each attribute. Let's start with the attribute *age*. We need to look at the distribution of *yes* and *no* tuples for each category of *age*. For the *age* category "youth," there are two *yes* tuples and three *no* tuples. For the category "middle_aged," there are four *yes* tuples and zero *no* tuples. For the category "senior," there are three *yes* tuples and two *no* tuples. Using Eq. (8.2), the expected information needed to classify a tuple in D if the tuples are partitioned according to *age* is

$$\begin{aligned} Info_{age}(D) &= \frac{5}{14} \times \left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right) \\ &+ \frac{4}{14} \times \left(-\frac{4}{4}\log_2\frac{4}{4}\right) \\ &+ \frac{5}{14} \times \left(-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}\right) \\ &= 0.694 \text{ bits.} \end{aligned}$$

Hence, the gain in information from such a partitioning would be

$$Gain(age) = Info(D) - Info_{age}(D) = 0.940 - 0.694 = 0.246 \text{ bits.}$$

Similarly, we can compute $Gain(income) = 0.029$ bits, $Gain(student) = 0.151$ bits, and $Gain(credit\_rating) = 0.048$ bits. Because *age* has the highest information gain among the attributes, it is selected as the splitting attribute. Node N is labeled with *age*, and branches are grown for each of the attribute's values. The tuples are then partitioned accordingly, as shown in Figure 8.5. Notice that the tuples falling into the partition for $age = middle\_aged$ all belong to the same class. Because they all belong to class *"yes,"* a leaf should therefore be created at the end of this branch and labeled *"yes."* The final decision tree returned by the algorithm was shown earlier in Figure

Lab exercises

1. Implement a decision tree using Information gain attribute selection measure on a suitable data set of your choice.
2. Consider Iris Data Set. Construct Decision tree with and without application of PCA. Does Decision tree perform better on application of PCA.

Additional exercises

1. Change the attribute selection measure to Gini and Gain ratio and observe the output of above decision tree construction implementation.

LAB NO: 11 **Date:**

MINIPROJECT-PAPER SELECTION

Objectives

1. **To familiarize with any Data mining application**

Introduction

Students should select the paper in data mining area from indexed journals.

[OBSERVATION SPACE – LAB 11]

PAPER TITLE:

ABSTRACT:

DATA SET USED:

LAB NO: 12 **Date:**

MINIPROJECT IMPLEMENTATION

[OBSERVATION SPACE – LAB 12]

WORKDONE:

REFERENCES

1. Jiawei Han and Micheline Kamber, “Data Mining- Concepts and Techniques”, 3rd Edition, Morgan Kaufmann Publishers, 2011.

2. Arun K. Pujari, ”Data Mining Techniques”, University press, 2006.

3. G.K. Gupta,”Introduction to data mining with Case Studies”,Easter Economy edition, Prentice Hall of India,2006.

4. Pang-Ning Tan,Michael Steinbach and Vipin Kumar,”Introduction to Data Mining” Pearson Education,2007.

5. Infosphere Documentation.

6. Rapid miner Documentation.