ignou
THE PEOPLE'S
UNIVERSITY

Indira Gandhi National Open University
School of Computer and Information
Sciences (SOCIS)

MCSL-223
Computer Networks and Data Mining Lab

Block Title

1

Indira Gandhi National Open University
School of Computer & Information Sciences

MCSL-223
COMPUTER NETWORKS
AND
DATA MINING LAB

LAB MANUAL

SECTION 1
COMPUTER NETWORKS LAB

SECTION 2
DATA MINING LAB

PROGRAMME DESIGN COMMITTEE

Prof. (Retd.) S.K. Gupta , IIT, Delhi

Prof. T.V. Vijay Kumar JNU, New Delhi

Prof. Ela Kumar, IGDTUW, Delhi

Prof. Gayatri Dhingra, GVMITM, Sonipat

Mr. Milind Mahajani, Impressico Business Solutions, New Delhi

Sh. Shashi Bhushan Sharma, Associate Professor, SOCIS, IGNOU

Sh. Akshay Kumar, Associate Professor, SOCIS, IGNOU

Dr. P. Venkata Suresh, Associate Professor, SOCIS, IGNOU

Dr. V.V. Subrahmanyam, Associate Professor, SOCIS, IGNOU

Sh. M.P. Mishra, Assistant Professor, SOCIS, IGNOU

Dr. Sudhansh Sharma, Assistant Professor, SOCIS, IGNOU

COURSE DESIGN COMMITTEE

Prof. T.V. Vijay Kumar, JNU, New Delhi

Dr. Rahul Johri, USICT, GGSIPU, New Delhi

Mr. Vinay Kumar Sharma, NVLI, IGNOU

Sh. Shashi Bhushan Sharma, Associate Professor, SOCIS, IGNOU

Sh. Akshay Kumar, Associate Professor, SOCIS, IGNOU

Dr. P. Venkata Suresh, Associate Professor, SOCIS, IGNOU

Dr. V.V. Subrahmanyam, Associate Professor, SOCIS, IGNOU

Sh. M.P. Mishra, Assistant Professor, SOCIS, IGNOU

Dr. Sudhansh Sharma, Assistant Professor, SOCIS, IGNOU

SOCIS FACULTY

Prof. P. Venkata Suresh, Director, SOCIS, IGNOU
Prof. V.V. Subrahmanyam, SOCIS, IGNOU
Dr. Akshay Kumar, Associate Professor, SOCIS, IGNOU
Dr. Naveen Kumar, Associate Professor, SOCIS, IGNOU (on EOL)
Dr. M.P. Mishra, Associate Professor, SOCIS, IGNOU
Dr. Sudhansh Sharma, Assistant Professor, SOCIS, IGNOU

BLOCK PREPARATION TEAM

Section-1 Computer Networks Lab

Section -2 Data Mining Lab

Prof. D. K. Tayal *(Content Editor)*
CSE, IGDTUW, New Delhi

Prof. K. Swathi *(Course Writer – Section-2)*
NRIIT, Vijayawada(Rural), Andhra Pradesh

Prof. V. V. Subrahmanyam
SOCIS, IGNOU, New Delhi

Course Coordinator: Dr. Akshay Kumar

Print Production

Sh. Tilak Raj, Assistant Registrar (Publication), MPDD

November, 2021

ISBN-

Further information on the Indira Gandhi National Open University courses may be obtained from the University's office at Maidan Garhi, New Delhi-110068.
Printed and published on behalf of the Indira Gandhi National Open University, New Delhi by MPDD, IGNOU.

COURSE INTRODUCTION

This laboratory course is in continuation with MCS-218 *"Data Communication and Computer Networks" and MCS-221 "Data Warehousing and Data Mining"*. This block is on practical problems of Computer Networks and Data Mining. This lab course is an attempt to upgrade and enhance your theoretical skills obtained from theory to a practical environment. This lab course consists of **two credits**.

This lab course consists of 2 sections and is organized as follows:

Section – 1 Computer Networks Lab Attempt all the problems given session-wise for 10 sessions.

Section – 2 Data Mining Lab This section will help to explore practical skills of Data Mining using WEKA. Practice all the examples given in this material and also attempt all the practical problems, session-wise for 10 sessions.

I wish you an eventful and interesting journey to the second semester laboratory world!

SECTION 1 COMPUTER NETWORKS LAB

Structure **Page No.**

1.0 INTRODUCTION

This is the lab course, wherein you will have the hands on experience. You have studied the support course material (MCS-218 Data Communication and Computer Networks). In this section, Computer Network Programming using an Open Source platform, Network Simpulator-3 (NS-3) is provided illustratively. A list of programming problems is also provided at the end for each session. You are also provided some example codes in the end for your reference (Appendix). Please go through the general guidelines and the program documentation guidelines carefully.

1.1 OBJECTIVES

After completing this lab course you will be able to:

- Explore and understand the features available in the NS-3 software
- Create wired and wireless connection
- Test and verify various routing/ communication protocols
- Test and verify user defined routing protocols
- Perform analysis and evaluation of various available protocols

1.2 GENERAL GUIDELINES

- You should attempt all problems/assignments given in the list session wise.
- You may seek assistance in doing the lab exercises from the concerned lab instructor. Since the assignments have credits, the lab instructor is obviously not expected to tell you how to solve these, but you may ask questions concerning a technical problem.
- For each program you should add comments above each function in the code, including the main function. This should also include a description of the function written.

- These descriptions should be placed in the comment block immediately above the relevant function source code.
- The comment block above the main function should describe the purpose of the program. Proper comments are to be provide where and when necessary in the programming.
- The output should also include clear network topology diagram.
- If two or more submissions from different students appear to be of the same origin (i.e. are variants of essentially the same program), none of them will be counted. You are strongly advised not to copy somebody else's work.
- It is your responsibility to create a separate directory to store all the programs, so that nobody else can read or copy.
- Observation book and Lab record are compulsory
- The list of the programs (list of programs given at the end, session-wise) is available to you in this lab manual. For each session, you must come prepare with the algorithms and the programs written in the Observation Book. You should utilize the lab hours for executing the programs, testing for various desired outputs and enhancements of the programs.
- As soon as you have finished a lab exercise, contact one of the lab instructor / in-charge in order to get the exercise evaluated and also get the signature from him/her on the Observation book.
- Completed lab assignments should be submitted in the form of a Lab Record in which you have to write the algorithm, program code along with comments and output for various inputs given.
- The total no. of lab sessions (3 hours each) are 10 and the list of assignments is provided session-wise. It is important to observe the deadline given for each assignment.

1.3 INTRODUCTION TO NS-3

The network simulator 2 or 3 is basically tools to design the network topologies of users' desire and test them. As a user, you can design various network designs and simulate them using the NS2/NS3. Various network simulators are available on the internet few are using GUI while others are purely command based. To make it more user friendly one of the open source version of the network simulator (also available in GUI form) i.e. GNS3 is used to show real like devices to realize the devices.

GNS3 is a very powerful tool for simulating network and its behaviour. As a learner you can design and extend your own designs to virtual topologies and real world networks. Using GNS-3 you can interact with your own computer network and analyse various aspects of it. This is an open source network

simulator known as GNS-3 (Graphical Network Simulator-3). This simulator uses an emulator which runs various CISCO based networking device images. This image includes switches, routers, bridges and other networking devices. The emulator is called Dynamips and helps in running standard IOS images. There are few limitations of Dynamips as this is just an emulator and cannot replace the real hardware products.

The Dynamips emulator is accessed and run with the help of a text based front end known as Dynagen. This helps in interacting with Dynamips using a OOP application programming interface (API). These APIs are used to communicate and perform various tasks in Dynamips like instance reload, reboot, loading devices etc.

However, the core NS-3 is a C++ based library and can be used using scripting also. The scripting also helps in working with the standard NS-3 tracefile format i.e. .pcap files for various applications. The new features or modules in NS-3 are implemented and complied in C++ and merged into the system. In our case the basic NS-3 distribution is more than enough to work upon and therefore we will be limiting ourselves to this only. We recommend you to install "*allinone*" distribution for simplicity in installation and can be installed with a simple *install* script.

Prerequisites: To run NS-3 you should have C++ compiler (g++ or other), Python (3.6 or above), CMake and a build system (make, xcode etc.).

1.4 BASIC FEATURES OF NS-3

There are multiple network simulation tools existing on the Internet. The outstanding features of the NS-3 are following:

1. NS-3 is modular in nature and can be combined with external libraries.
2. NS-3 can be used on Linux, MacOs, BSD and Windows (using linux virtual machines).
3. NS-3 is an open source software product and project strive to remain open source and provide researchers and developers to contribute to the project.

As mentioned this is an open source project and hence you are not required to pay any amount to use or to implement this library at your own. It is dependent on the research and community to be maintained and further developed.

1.5 INSTALLING NS-3

Now, Let's talk about the download and installation of the network simulator software. The NS-3 is managed as a project by NS-3 consortium and available at the following link: www.nsnam.org.

You can download NS-3 freely from NS-3 website and following the installation guidelines, can easily install on different OS platforms like Mac OS, Linux and Windows. Originally, the NS-3 is based on the Linux environment, but if you have Windows environment, you can use cygwin to create a running environment for Linux based applications on windows as this provides the necessary environment for such applications. You can also use other hypervisor tools to create Linux environment in Windows and then install NS-3.

For using NS-3 all-in-one installation file GCC and other dependencies must be installed on the system. These dependencies can be found on the NS-3 official website.

Prerequisite	**Package/version**
C++ compiler	clang++ or g++ (g++ version 8 or greater)
Python	python3 version >=3.6
CMake	cmake version >=3.10
Build system	make, ninja, xcodebuild (XCode)
Git	any recent version (to access *ns-3* from GitLab.com)
tar	any recent version (to unpack an ns-3 release)
bunzip2	any recent version (to uncompress an *ns-3* release)

Following are the steps involved in installation of NS-3

1. Download and unzip tar file from official website.
2. Run the *build.py* script file as

   ./buid.py --enable-examples --enable-tests

3. Check for list of Modules built and Modules not built.
4. After successfully build operation you should see different files and directories like source directory *src,* execution script *waf* and one script directory *scratch.* It should also contain the *examples* and *tutorials* files.

(A detailed description is provided on this link for installation of NS-3 on Windows https://www.nsnam.org/wiki/Installation#Windows).

At first you will see building of netanim animator build then pybindgen and finally NS-3. Now, NS-3 is installed and ready to be used you can run few examples on the system and try to learn the working of the system. Before running actual program you should run test programs to verify the running environment. To run the test programs you can run ./test.py --no-build command see if the systems pass all the tests. Once it passes all tests, you are ready to run your own code.

1.6 EXAMPLES

Let us now try to run an example on the recently installed system. Create a file with any name and save it with an extension as *.cc* or you can just go to examples/tutorials directory and see *first.cc* example file.

To run the first example, copy the *first.cc* file to scratch directory and run the following commands in the NS-2 terminal (cygwin) from the parent directory

./waf –run first

If nothing goes wrong, the program is compiled and run.

Argument Passing: NS-3 also supports the command line options and can pass the arguments to the program using command line. The following example with file *second.cc* program shows the argument value 3 given to *nCsma* variable.

./waf --run "second --nCsma=3"

Error Compilation: NS-3 by default treat the warnings as errors (enabling –Werror option) which is a good sign for professional but a little more exercise for beginners. You can disable this option by editing the *waf-tools/cflags.py* and change the following line:

self.warnings_flags = [['-Wall'], ['-Werror'], ['-Wextra']]

to the following line

self.warnings_flags = [['-Wall'], ['-Wextra']]

Now, you need to run the following commands.

./waf configure
./waf build

Once you are aware about few of the basic commands, you can see the examples given in the documentation of the NS-3. The link for the documentation is given below. A sample program is provided at the end of this file for tracing packets.

https://www.nsnam.org/doxygen/tcp-bbr-example_8cc_source.html

1.7 TERMS AND CONCEPTS

Before we proceed further, let's understand and map few terms to the actual networking concepts with the NS-3 environment to simplify the coding.

Node: A basic computing device is termed as a node (computer also) in NS-3. This is also abstracted as node class in NS-3. It also has methods for its management and representations of the device.

Application: The software with interacts with the operating systems and does the specified job is termed as an application. NS-3 refers to application as a user program which generates an activity which is further simulated. The Application class provides various associated methods to managing the user level applications.

Channel: Channels are basically the media of communication; here channel class manages the communication subnetwork objects and connecting nodes in the network. The three different channels used by us are CsmaChannel, PointToPointChannel and WifiChannel.

Net Device: A net device is usually an abstraction of both simulated hardware and software driver. After installation, the net device enables the node to communicate with other nodes using channels (in simulations).

Figure 1, shows the basic data flow model in simulator using ns-3. Applications talk with protocol stalks using APIs which are then used to communicate using NetDevices via channels.

There are many other terms and concepts which you can refer online on at the following link:

https://www.nsnam.org/docs/release/3.35/tutorial/html/conceptual-overview.html#key-abstractions

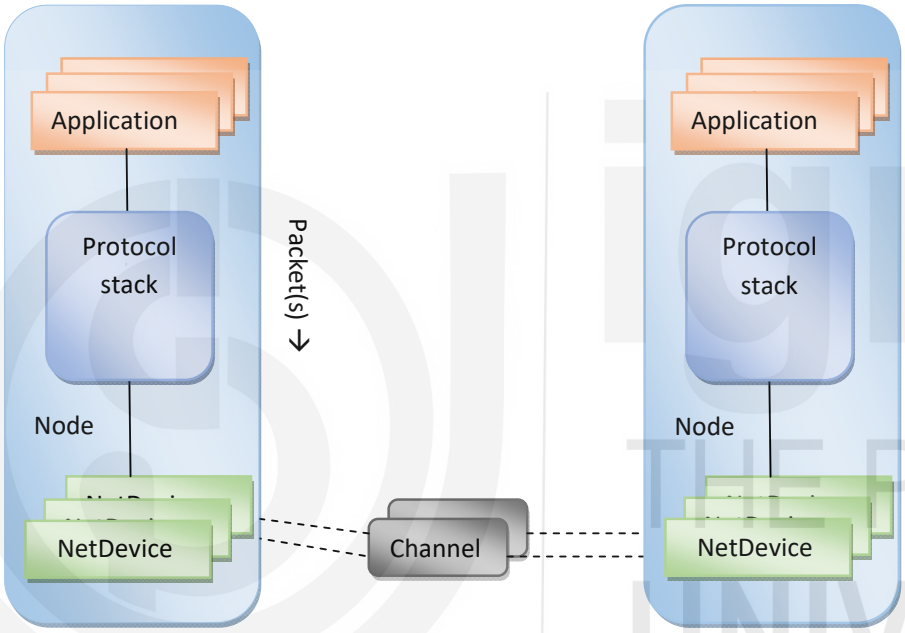


Figure 1: Basic data flow model in NS-3

1.8 LIST OF LAB ASSIGNMENTS – SESSION WISE

Let us try solving a different kind of a problem now. See the following given problems and try to implement these with the help of NS-3 simulator.

Session-1

1. Create a simple point to point network topology using two nodes.
2. Create a UdpClient and UdpServer nodes and communicate at a fixed data rate.

Session - 2

3. Measure the throughput (end to end) while varying latency in the network created in Session -1.

4. Create a simple network topology having two client node on left side and two server nodes on the right side. Both clients are connected with another node n1. Similarly both server node connecting to node n2. Also connect node n1 and n2 thus forming a dumbbell shape topology. Use point to point link only.

Session - 3

5. Install a TCP socket instance connecting either of the client node with either of the server node in session 2's network topology.

6. Install a TCP socket instance connecting other remaining client node with the remaining server node in session 2's network topology.

7. Start TCP application and monitor the packet flow.

Session - 4

8. Take three nodes n1, n2 and n3 and create a wireless mobile ad-hoc network.
9. Install the optimized Link State Routing protocol on these nodes.

Session – 5

10. Create a UDP client on a node n1 and a UDP server on a node n2.
11. Send packets to node n2 from node n1 and plot the number of bytes received with respect to time at node n2.
12. Show the pcap traces at node n2's WiFi interface.

Session – 6

13. Use 2 nodes to setup a wireless ad-hoc network where nodes are placed at a fixed distance in a 3D plane.

14. Install UDP server and Client at these two nodes.

15. Setup a CBR transmission between these nodes.

Session – 7

16. Setup 4 nodes, two TCP client and server pair and two UDP client and server pair.
17. Send packets to respective clients from both the servers.
18. Monitor the traffic for both the pair and plot the no. of bytes received.

Session – 8

19. Use the setup made in session 2 and monitor the traffic flow, plot the packets received.
20. Start the TCP application at Time 1 second.
21. After 20 seconds, start the UDP application at Rate1 which clogs the half of the dumbbell bridge capacity.

22. Using ns-3 tracing mechanism, plot the changes in the TCP window size over the time.

Session – 9

23. In the last session 8, Increase the UDP rate at 30 second to Rate2 such that it clogs whole of the dumbbell bridge capacity.
24. Use MatPlotlLib or GNUPlot to visualize cwnd vs time, also mention Rate1 and Rate2.

Session -10

25. Create a point to pint network between two nodes with the following parameters.

    Link bandwidth between the two nodes. Default is 5 Mbps.
    One way delay of the link. Default is 5 milliseconds.
    Loss rate of packets on the link. Default is 0.000001. (This covers losses other than those that occur due to buffer drops at node0.)
    Queue size of the buffer at node 0. Default is 10 packets.
    Simulation time. Default is 10 seconds.

    Calculate the average TCP throughput at the receiver using Wireshark application for packet capturing.

1.9 WEB REFERENCES

2. https://www.cse.iitb.ac.in/~mythili/teaching/cs224m_autumn2017/tcpsimpa/index.html
3. http://intronetworks.cs.luc.edu/current/html/ns3.html
4. https://www.nsnam.org/doxygen
5. https://www.nsnam.org/doxygen/csma-bridge_8cc_source.html
6. https://www.nsnam.org/docs/release/3.35/tutorial/html/conceptual-overview.html#key-abstractions
7. https://www.nsnam.org/wiki/Installation#Windows
8. https://www.wireshark.org/docs/wsug_html_chunked/ChapterIO.html

Appendix

Sample programs available online for different applications using NS-3

CODE-1: Source- https://www.nsnam.org/doxygen/traceroute-example_8cc_source.html

```
/* Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; */
/*
* Copyright (c) 2019 Ritsumeikan University, Shiga, Japan
*
* This program is free software; you can redistribute it *and/or modify it under the terms of the GNU General Public *License version 2 as published by the Free Software *Foundation;
*
* This program is distributed in the hope that it will be *useful, but WITHOUT ANY WARRANTY; without even the implied *warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR *PURPOSE. See the GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public *License along with this program; if not, write to the Free *Software Foundation, Inc., 59 Temple Place, Suite 330, *Boston, MA 02111-1307 USA
*
* Author: Alberto Gallegos Ramonet <ramonet@fc.ritsumei.ac.jp>
*
*
* TraceRoute application example using AODV routing protocol.
*
*
*/
#include "ns3/aodv-module.h"
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/wifi-module.h"
#include "ns3/v4traceroute-helper.h"
#include <iostream>
#include <cmath>
using namespace ns3;
class TracerouteExample
{
public:
TracerouteExample ();
bool Configure (int argc, char **argv);
void Run ();
void Report (std::ostream & os);
private:
// parameters
uint32_t size;
double step;
double totalTime;
bool pcap;
```

```
bool printRoutes;
NodeContainer nodes;
NetDeviceContainer devices;
Ipv4InterfaceContainer interfaces;
private:
void CreateNodes ();
void CreateDevices ();
void InstallInternetStack ();
void InstallApplications ();
};
int main (int argc, char **argv)
{
TracerouteExample test;
if (!test.Configure (argc, argv))
{
NS_FATAL_ERROR ("Configuration failed. Aborted.");
}
test.Run ();
test.Report (std::cout);
return 0;
}
//-----------------------------------------------------------------------------
TracerouteExample::TracerouteExample ()
: size (10),
step (50),
totalTime (100),
pcap (false),
printRoutes (false)
{
}
bool
TracerouteExample::Configure (int argc, char **argv)
{
// Enable AODV logs by default. Comment this if too noisy
// LogComponentEnable("AodvRoutingProtocol", LOG_LEVEL_ALL);
SeedManager::SetSeed (12345);
CommandLine cmd (__FILE__);
cmd.AddValue ("pcap", "Write PCAP traces.", pcap);
cmd.AddValue ("printRoutes", "Print routing table dumps.", printRoutes);
cmd.AddValue ("size", "Number of nodes.", size);
cmd.AddValue ("time", "Simulation time, s.", totalTime);
cmd.AddValue ("step", "Grid step, m", step);
cmd.Parse (argc, argv);
return true;
}
void
TracerouteExample::Run ()
{
CreateNodes ();
CreateDevices ();
InstallInternetStack ();
InstallApplications ();
std::cout << "Starting simulation for " << totalTime << " s ...\n";
Simulator::Stop (Seconds (totalTime));
Simulator::Run ();
Simulator::Destroy ();
```

```
}
void
TracerouteExample::Report (std::ostream &)
{
}
void
TracerouteExample::CreateNodes ()
{
std::cout << "Creating " << (unsigned)size << " nodes " << step << " m apart.\n";
nodes.Create (size);
// Name nodes
for (uint32_t i = 0; i < size; ++i)
{
std::ostringstream os;
os << "node-" << i;
Names::Add (os.str (), nodes.Get (i));
}
// Create static grid
MobilityHelper mobility;
mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
"MinX", DoubleValue (0.0),
"MinY", DoubleValue (0.0),
"DeltaX", DoubleValue (step),
"DeltaY", DoubleValue (0),
"GridWidth", UintegerValue (size),
"LayoutType", StringValue ("RowFirst"));
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (nodes);
}
void
TracerouteExample::CreateDevices ()
{
WifiMacHelper wifiMac;
wifiMac.SetType ("ns3::AdhocWifiMac");
YansWifiPhyHelper wifiPhy;
YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
wifiPhy.SetChannel (wifiChannel.Create ());
WifiHelper wifi;
wifi.SetRemoteStationManager ("ns3::ConstantRateWifiManager", "DataMode",
StringValue ("OfdmRate6Mbps"), "RtsCtsThreshold", UintegerValue (0));
devices = wifi.Install (wifiPhy, wifiMac, nodes);
if (pcap)
{
wifiPhy.EnablePcapAll (std::string ("aodv"));
}
}
void
TracerouteExample::InstallInternetStack ()
{
AodvHelper aodv;
// you can configure AODV attributes here using aodv.Set(name, value)
InternetStackHelper stack;
stack.SetRoutingHelper (aodv); // has effect on the next Install ()
stack.Install (nodes);
Ipv4AddressHelper address;
address.SetBase ("10.0.0.0", "255.0.0.0");
```

```
interfaces = address.Assign (devices);
if (printRoutes)
{
Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper>
("aodv.routes", std::ios::out);
aodv.PrintRoutingTableAllAt (Seconds (8), routingStream);
}
}
void
TracerouteExample::InstallApplications ()
{
V4TraceRouteHelper traceroute (Ipv4Address ("10.0.0.10")); //size - 1
traceroute.SetAttribute ("Verbose", BooleanValue (true));
ApplicationContainer p = traceroute.Install (nodes.Get (0));
// Used when we wish to dump the traceroute results into a file
//Ptr<OutputStreamWrapper> printstrm = Create<OutputStreamWrapper> ("mytrace",
std::ios::out);
//traceroute.PrintTraceRouteAt(nodes.Get(0),printstrm);
p.Start (Seconds (0));
p.Stop (Seconds (totalTime) - Seconds (0.001));
}
```

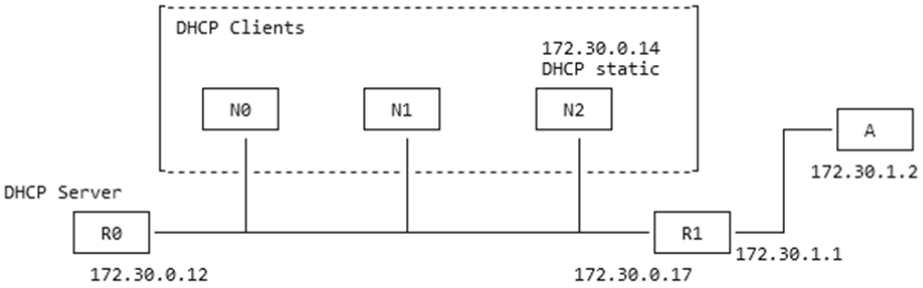
CODE-2: Source- https://www.nsnam.org/doxygen/dhcp-example_8cc_source.html

---

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
* Copyright (c) 2011 UPB
* Copyright (c) 2017 NITK Surathkal
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License version 2 as
* published by the Free Software Foundation;
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*
* Author: Radu Lupu <rlupu@elcom.pub.ro>
* Ankit Deepak <adadeepak8@gmail.com>
* Deepti Rajagopal <deeptir96@gmail.com>
*
*/
/*
```

```
Network layout:

R0 is a DHCP server. The DHCP server announced R1 as the default router.
Nodes N1 will send UDP Echo packets to node A.
```




```
Things to notice:
1) The routes in A are manually set to have R1 as the default router,
   just because using a dynamic outing in this example is an overkill.
2) R1's address is set statically though the DHCP server helper interface.
   This is useful to prevent address conflicts with the dynamic pool.
   Not necessary if the DHCP pool is not conflicting with static addresses.
3) N2 has a dynamically-assigned, static address (i.e., a fixed address assigned via DHCP).
```

```
*
*/
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-apps-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("DhcpExample");
int
main (int argc, char *argv[])
{
CommandLine cmd (__FILE__);
bool verbose = false;
bool tracing = false;
cmd.AddValue ("verbose", "turn on the logs", verbose);
cmd.AddValue ("tracing", "turn on the tracing", tracing);
cmd.Parse (argc, argv);
// GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true));
if (verbose)
{
LogComponentEnable ("DhcpServer", LOG_LEVEL_ALL);
LogComponentEnable ("DhcpClient", LOG_LEVEL_ALL);
LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
}
Time stopTime = Seconds (20);
NS_LOG_INFO ("Create nodes.");
NodeContainer nodes;
NodeContainer router;
nodes.Create (3);
router.Create (2);
NodeContainer net (nodes, router);
NS_LOG_INFO ("Create channels.");
CsmaHelper csma;
```

```
csma.SetChannelAttribute ("DataRate", StringValue ("5Mbps"));
csma.SetChannelAttribute ("Delay", StringValue ("2ms"));
csma.SetDeviceAttribute ("Mtu", UintegerValue (1500));
NetDeviceContainer devNet = csma.Install (net);
NodeContainer p2pNodes;
p2pNodes.Add (net.Get (4));
p2pNodes.Create (1);
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);
InternetStackHelper tcpip;
tcpip.Install (nodes);
tcpip.Install (router);
tcpip.Install (p2pNodes.Get (1));
Ipv4AddressHelper address;
address.SetBase ("172.30.1.0", "255.255.255.0");
Ipv4InterfaceContainer p2pInterfaces;
p2pInterfaces = address.Assign (p2pDevices);
// manually add a routing entry because we don't want to add a dynamic routing
Ipv4StaticRoutingHelper ipv4RoutingHelper;
Ptr<Ipv4> ipv4Ptr = p2pNodes.Get (1)->GetObject<Ipv4> ();
Ptr<Ipv4StaticRouting> staticRoutingA = ipv4RoutingHelper.GetStaticRouting (ipv4Ptr);
staticRoutingA->AddNetworkRouteTo (Ipv4Address ("172.30.0.0"), Ipv4Mask ("/24"),
Ipv4Address ("172.30.1.1"), 1);
NS_LOG_INFO ("Setup the IP addresses and create DHCP applications.");
DhcpHelper dhcpHelper;
// The router must have a fixed IP.
Ipv4InterfaceContainer fixedNodes = dhcpHelper.InstallFixedAddress (devNet.Get (4),
Ipv4Address ("172.30.0.17"), Ipv4Mask ("/24"));
// Not really necessary, IP forwarding is enabled by default in IPv4.
fixedNodes.Get (0).first->SetAttribute ("IpForward", BooleanValue (true));
// DHCP server
ApplicationContainer dhcpServerApp = dhcpHelper.InstallDhcpServer (devNet.Get (3),
Ipv4Address ("172.30.0.12"),
Ipv4Address ("172.30.0.0"), Ipv4Mask ("/24"),
Ipv4Address ("172.30.0.10"), Ipv4Address ("172.30.0.15"),
Ipv4Address ("172.30.0.17"));
// This is just to show how it can be done.
DynamicCast<DhcpServer> (dhcpServerApp.Get (0))->AddStaticDhcpEntry (devNet.Get
(2)->GetAddress (), Ipv4Address ("172.30.0.14"));
dhcpServerApp.Start (Seconds (0.0));
dhcpServerApp.Stop (stopTime);
// DHCP clients
NetDeviceContainer dhcpClientNetDevs;
dhcpClientNetDevs.Add (devNet.Get (0));
dhcpClientNetDevs.Add (devNet.Get (1));
dhcpClientNetDevs.Add (devNet.Get (2));
ApplicationContainer dhcpClients = dhcpHelper.InstallDhcpClient (dhcpClientNetDevs);
dhcpClients.Start (Seconds (1.0));
dhcpClients.Stop (stopTime);
UdpEchoServerHelper echoServer (9);
ApplicationContainer serverApps = echoServer.Install (p2pNodes.Get (1));
serverApps.Start (Seconds (0.0));
serverApps.Stop (stopTime);
```

```
UdpEchoClientHelper echoClient (p2pInterfaces.GetAddress (1), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (100));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
ApplicationContainer clientApps = echoClient.Install (nodes.Get (1));
clientApps.Start (Seconds (10.0));
clientApps.Stop (stopTime);
Simulator::Stop (stopTime + Seconds (10.0));
if (tracing)
{
csma.EnablePcapAll ("dhcp-csma");
pointToPoint.EnablePcapAll ("dhcp-p2p");
}
NS_LOG_INFO ("Run Simulation.");
Simulator::Run ();
Simulator::Destroy ();
NS_LOG_INFO ("Done.");
}
```

CODE-3: Source- https://www.nsnam.org/doxygen/csma-bridge_8cc_source.html

---

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
/*
* This program is free software; you can redistribute it *and/or modify it under the terms of the GNU General Public *License version 2 as  published by the Free Software *Foundation;
*
* This program is distributed in the hope that it will be *useful, but WITHOUT ANY WARRANTY; without even the implied *warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR *PURPOSE. See the GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public *License along with this program; if not, write to the Free *Software Foundation, Inc., 59 Temple Place, Suite 330, *Boston, MA 02111-1307 USA
*/
// Network topology
//
// n0     n1
// |      |
// ----------
// | Switch |
// ----------
// |      |
// n2     n3
//
// - CBR/UDP flows from n0 to n1 and from n3 to n0
// - DropTail queues
// - Tracing of queues and packet receptions to file "csma-bridge.tr"
#include <iostream>
#include <fstream>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/bridge-module.h"
```

```
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
using namespace ns3;
NS_LOG_COMPONENT_DEFINE ("CsmaBridgeExample");
int
main (int argc, char *argv[])
{
//
// Users may find it convenient to turn on explicit debugging
// for selected modules; the below lines suggest how to do this
//
#if 0
LogComponentEnable ("CsmaBridgeExample", LOG_LEVEL_INFO);
#endif
//
// Allow the user to override any of the defaults and the above Bind() at
// run-time, via command-line arguments
//
CommandLine cmd (__FILE__);
cmd.Parse (argc, argv);
//
// Explicitly create the nodes required by the topology (shown above).
//
NS_LOG_INFO ("Create nodes.");
NodeContainer terminals;
terminals.Create (4);
NodeContainer csmaSwitch;
csmaSwitch.Create (1);
NS_LOG_INFO ("Build Topology");
CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", DataRateValue (5000000));
csma.SetChannelAttribute ("Delay", TimeValue (MilliSeconds (2)));
// Create the csma links, from each terminal to the switch
NetDeviceContainer terminalDevices;
NetDeviceContainer switchDevices;
for (int i = 0; i < 4; i++)
{
NetDeviceContainer link = csma.Install (NodeContainer (terminals.Get (i), csmaSwitch));
terminalDevices.Add (link.Get (0));
switchDevices.Add (link.Get (1));
}
// Create the bridge netdevice, which will do the packet switching
Ptr<Node> switchNode = csmaSwitch.Get (0);
BridgeHelper bridge;
bridge.Install (switchNode, switchDevices);
// Add internet stack to the terminals
InternetStackHelper internet;
internet.Install (terminals);
// We've got the "hardware" in place. Now we need to add IP addresses.
//
NS_LOG_INFO ("Assign IP Addresses.");
Ipv4AddressHelper ipv4;
ipv4.SetBase ("10.1.1.0", "255.255.255.0");
ipv4.Assign (terminalDevices);
//
// Create an OnOff application to send UDP datagrams from node zero to node 1.
```

```
//
NS_LOG_INFO ("Create Applications.");
uint16_t port = 9; // Discard port (RFC 863)
OnOffHelper onoff ("ns3::UdpSocketFactory",
Address (InetSocketAddress (Ipv4Address ("10.1.1.2"), port)));
onoff.SetConstantRate (DataRate ("500kb/s"));
ApplicationContainer app = onoff.Install (terminals.Get (0));
// Start the application
app.Start (Seconds (1.0));
app.Stop (Seconds (10.0));
// Create an optional packet sink to receive these packets
PacketSinkHelper sink ("ns3::UdpSocketFactory",
Address (InetSocketAddress (Ipv4Address::GetAny (), port)));
app = sink.Install (terminals.Get (1));
app.Start (Seconds (0.0));
//
// Create a similar flow from n3 to n0, starting at time 1.1 seconds
//
onoff.SetAttribute ("Remote",
AddressValue (InetSocketAddress (Ipv4Address ("10.1.1.1"), port)));
app = onoff.Install (terminals.Get (3));
app.Start (Seconds (1.1));
app.Stop (Seconds (10.0));
app = sink.Install (terminals.Get (0));
app.Start (Seconds (0.0));
NS_LOG_INFO ("Configure Tracing.");
//
// Configure tracing of all enqueue, dequeue, and NetDevice receive events.
// Trace output will be sent to the file "csma-bridge.tr"
//
AsciiTraceHelper ascii;
csma.EnableAsciiAll (ascii.CreateFileStream ("csma-bridge.tr"));
//
// Also configure some tcpdump traces; each interface will be traced.
// The output files will be named:
// csma-bridge-<nodeId>-<interfaceId>.pcap
// and can be read by the "tcpdump -r" command (use "-tt" option to
// display timestamps correctly)
//
csma.EnablePcapAll ("csma-bridge", false);
//
// Now, do the actual simulation.
//
NS_LOG_INFO ("Run Simulation.");
Simulator::Run ();
Simulator::Destroy ();
NS_LOG_INFO ("Done.");
}
```

SECTION 2 DATA MINING LAB

Structure **Page Nos.**

2.0 INTRODUCTION

You have studied the course material (MCS-221 Data Warehousing and Data Mining). This is the lab course, wherein you will get the hands on experience on Data Mining tasks using WEKA open source software. Along with some of the examples discussed in this course material, a separate list of practical problems are given sessionwise in section 2.13. Please practice all the examples and later on attempt all the practical problems, session-wise (10 sessions). Also, go through the general guidelines given in section 2.12 carefully.

2.1 OBJECTIVES

After going through this practical course, you will be able to:

- Understand how to handle data mining tasks using a data mining toolkit (such as open source WEKA)
- Understand the various kinds of options available in WEKA
- Understand various data sets and data preprocessing
- Demonstrate the working of algorithms for data mining tasks such association rule mining, classification, clustering and regression
- Exercise the data mining techniques with varied input values for different parameters
- Emphasize hands-on experience working with all real data sets.
- Ability to add mining algorithms as a component to the exiting tools
- Ability to apply mining techniques for realistic data.

2.2 INTRODUCTION TO WEKA

WEKA is a data mining/machine learning application and is being developed by Waikato University in New Zealand. WEKA is an open-source tool designed and developed by the scientists/researchers at the University of Waikato, New Zealand. WEKA stands for Waikato Environment for Knowledge Analysis. It is developed by the international scientific community and distributed under the free GNU GPL license.

WEKA is fully developed in Java. It provides integration with the SQL database using Java Database connectivity. It provides many machine learning algorithms to implement data mining tasks. These algorithms can either be used directly using the WEKA tool or can be used with other applications using Java programming language.

It provides a lot of tools for data preprocessing, classification, clustering, regression analysis, association rule creation, feature extraction, and data visualization. It is a powerful tool that supports the development of new algorithms in machine learning.

With WEKA, the machine learning algorithms are readily available to the users. The ML specialists can use these methods to extract useful information from high volumes of data. Here, the specialists can create an environment to develop new machine learning methods and implement them on real data.

WEKA is used by machine learning and applied sciences researchers for learning purposes. It is an efficient tool for carrying out many data mining tasks.

Advantages of WEKA include:

- Free availability under the GNU General Public License.
- Portability, since it is fully implemented in the Java programming language and thus runs on almost any modern computing platform.
- A comprehensive collection of data preprocessing and modeling techniques.
- Ease of use due to its graphical user interfaces.

2.3 LATEST VERSION AND DOWNLOADS

There are two versions of WEKA: WEKA 3.8 is the latest stable version and WEKA 3.9 is the development version.

The stable version receives only bug fixes and feature upgrades that do not break compatibility with its earlier releases, while the development version may receive new features that break compatibility with its earlier releases.

WEKA 3.8 and 3.9 feature a package management system that makes it easy for the WEKA community to add new functionality to WEKA. The package management system requires an internet connection in order to download and install packages.

Stable version

WEKA 3.8 is the latest stable version of WEKA. This branch of WEKA only receives bug fixes and upgrades that do not break compatibility with earlier 3.8 releases, although major new features may become available in packages. There are different options for downloading and installing it on your system:

Windows

Use https://sourceforge.net/projects/weka/files/weka-3-8/3.8.5/weka-3-8-5-azul-zulu-windows.exe/download?use_mirror=nchc to download a self-extracting executable for 64-bit Windows that includes Azul's 64-bit OpenJDK Java VM 11 (weka-3-8-5-azul-zulu-windows.exe; 124.6 MB)

This executable will install WEKA in your Program Menu. Launching via the Program Menu or shortcuts will automatically use the included JVM to run WEKA.

Linux

Use https://sourceforge.net/projects/weka/files/weka-3-8/3.8.5/weka-3-8-5-azul-zulu-arm-osx.dmg/download?use_mirror=nchc to download a zip archive for Linux that includes Azul's 64-bit OpenJDK Java VM 11 (weka-3-8-5-azul-zulu-linux.zip; 137.4 MB)

First unzip the zip file. This will create a new directory called weka-3-8-5. To run Weka, change into that directory and type:
./weka.sh

WEKA packages

There is a list of packages for WEKA that can be installed using the built-in package manager. Javadoc for a package is available at https://weka.sourceforge.io/doc.packages/ followed by the name of the package.

Requirements

The latest official releases of WEKA require Java 8 or later. Note that if you are using Windows and your computer has a display with high pixel density (HiDPI), you may need to use Java 9 or later to avoid problems with inappropriate scaling of WEKA's graphical user interfaces.

2.4 INSTALLATION OF WEKA

1. Download the software WEKA from: https://sourceforge.net/projects/weka/files/weka-3-8/3.8.5/weka-3-8-5-azul-zulu-windows.exe/download?use_mirror=nchc for WINDOWS operating system. Check the configuration of the computer system and download the stable version of WEKA (currently 3.8).

2. After successful download, open the file location and double click on the downloaded file. The Step Up wizard will appear. Click on Next.

3. The License Agreement terms will open. Read it thoroughly and click on "I Agree".

4. According to your requirements, select the components to be installed. Full component installation is recommended. Click on Next.

5. Select the destination folder and Click on Next.

6. Then, Installation will start.

7. After Installation is complete, WEKA Tool and Explorer window opens as shown in Figure 1.

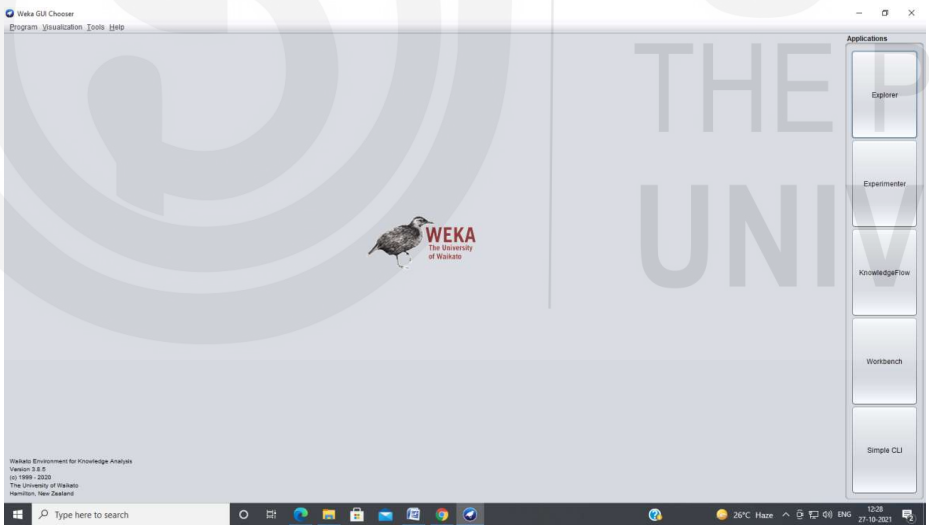


Figure 1: WEKA GUI Interface

The GUI of WEKA gives five options: Explorer, Experimenter, Knowledge flow, Workbench, and Simple CLI. Let us understand each of these individually.

1. **Simple CLI** is WEKA Shell with command line and output. With "help", the overview of all the commands can be seen. Simple CLI offers access to all classes such as classifiers, clusters, and filters, etc.
2. **Explorer** is an environment to discover the data. The WEKA Explorer window show different tabs starting with preprocess. Initially, the

preprocess tab is active, as first the data set is preprocessed before applying algorithms to it and explored the dataset.

3. **Experimenter** is an environment to make experiments and statistical tests between learning schemes. The WEKA experimenter button allows the users to create, run, and modify different schemes in one experiment on a dataset. The experimenter has 2 types of configuration: ***Simple*** and ***Advanced***. Both configurations allow users to run experiments locally and on remote computers.
4. **KnowledgeFlow** is a Java-Beans based interface for tuning and machine learning experiments. Knowledge flow shows a graphical representation of WEKA algorithms. The user can select the components and create a workflow to analyze the datasets. The data can be handled by batch-wise or incrementally. Parallel workflows can be designed and each will run in a separate thread.The different components available are Datasources, Datasavers, Filters, Classifiers, Clusters, Evaluation, and Visualization.
5. WEKA has workbench module which contains all the GUI's in a single window.

2.5 DATASETS

Below are some sample WEKA data sets available in ***.arff*** format.

- airline.arff
- breast-cancer.arff
- contact-lens.arff
- cpu.arff
- cpu.with-vendor.arff
- credit-g.arff
- diabetes.arff
- glass.arff
- hypothyroid.arff
- ionospehre.arff
- iris.2D.arff
- iris.arff
- labor.arff
- reutersCorn-train.arff
- reutersCorn-test.arff
- reutersGrain-train.arff
- reutersGrain-test.arff
- segment-challenge.arff
- segment-test.arff
- soybean.arff
- supermarket.arff
- unbalanced.arff
- vote.arff
- weather.numeric.arff
- weather.nominal.arff

The WEKA datasets can be explored from the **"C:\Program Files\Weka-3-8\data"** link. The datasets are in **.** ***arff*** format.

Miscellaneous Collections of Datasets

- A jarfile containing 37 classification problems originally obtained from the UCI repository of machine learning datasets are available at http://archive.ics.uci.edu/ml/index.php .
- A jarfile containing 37 regression problems obtained from various sources (datasets-numeric.jar available at https://sourceforge.net/projects/weka/files/datasets/datasets-numeric/datasets-numeric.jar/download?use_mirror=netix .
- A jarfile containing 6 agricultural datasets obtained from agricultural researchers in New Zealand (agridatasets.jar, 31,200 Bytes).
- A jarfile containing 30 regression datasets collected by Professor Luis Torgo (regression-datasets.jar, 10,090,266 Bytes).
- A gzip'ed tar containing UCI ML at http://archive.ics.uci.edu/ml/index.php and UCI KDD datasets at https://kdd.ics.uci.edu/ .
- A gzip'ed tar containing StatLib datasets at https://sourceforge.net/projects/weka/files/datasets/UCI%20and%20StatLib/statlib-20050214.tar.gz/download?use_mirror=kumisystems
- A gzip'ed tar containing ordinal, real-world datasets donated by Professor Arie Ben David at https://sourceforge.net/projects/weka/files/datasets/regression-datasets/datasets-arie_ben_david.tar.gz/download?use_mirror=pilotfiber
- A zip file containing 19 multi-class (1-of-n) text datasets donated by Dr George Forman available at https://sourceforge.net/projects/weka/files/datasets/text-datasets/19MclassTextWc.zip/download?use_mirror=netactuate&download = (19MclassTextWc.zip, 14,084,828 Bytes).
- A bzip'ed tar file containing the Reuters21578 dataset split into separate files according to the ModApte split reuters21578-ModApte.tar.bz2, 81,745,032 Bytes.
- A zip file containing 41 drug design datasets formed using the Adriana.Code software donated by Dr Mehmet Fatih Amasyali at https://sourceforge.net/projects/weka/files/datasets/Drug%20design%20datasets/Drug-datasets.zip/download?use_mirror=netix&use_mirror=internode .
- A zip file containing 80 artificial datasets generated from the Friedman function donated by Dr. M. Fatih Amasyali (Yildiz Technical Unversity) (Friedman-datasets.zip, 5,802,204 Bytes)

- A zip file containing a new, image-based version of the classic iris data, with 50 images for each of the three species of iris. The images have size 600x600. Please see the ARFF file for further information (iris_reloaded.zip, 92,267,000 Bytes). After expanding into a directory using your jar utility (or an archive program that handles tar-archives/zip files in case of the gzip'ed tars/zip files), these datasets may be used with WEKA.

2.6 FEATURES OF WEKA EXPLORER

In this section we will learn about the features of the WEKA Explorer.

2.6.1 Dataset

A dataset is made of items. It represents an object **for example:** in the marketing database, it will represent customers and products. The datasets are described by attributes. The dataset contains data tuples in a database. A dataset has attributes that can be nominal, numeric, or string. In Weka, the dataset is represented by **weka.core.Instances** class.Representation of dataset with 5 examples:

@data
sunny, FALSE,85,85,no
sunny, TRUE,80,90,no
overcast, FALSE,83,86,yes
rainy, FALSE,70,96,yes
rainy, FALSE,68,80,yes

Attribute and its Types

An attribute is a data field representing the characteristic of a data object. For example, in a customer database, the attributes will be customer_id, customer_email, customer_address, etc. Attributes have different types:
(i) Nominal Attributes: Attribute which relates to a name and has predefined values such as color, weather. These attributes are called ***categorical attributes***. These attributes do not have any order and their values are also called enumerations.
@attribute outlook {sunny, overcast, rainy}: declaration of the nominal attribute.
(ii) Binary Attributes: These attributes represent only values 0 and 1. These are the type of nominal attributes with only 2 categories. These attributes are also called Boolean.
(iii) Ordinal Attributes: The attributes which preserve some order or ranking amongst them are ordinal attributes. Successive values cannot be predicted but only order is maintained. **Example:** size, grade, etc.
(iv) Numeric Attributes: Attributes representing measurable quantities are numeric attributes. These are represented by real numbers or integers. **Example:** temperature, humidity.
@attribute humidity real: declaration of a numeric attribute
(v) String Attributes: These attributes represent a list of characters represented in double-quotes.

2.6.2 ARFF Data format

WEKA works on the ARFF file for data analysis. ARFF stands for Attribute Relation File Format. It has 3 sections: relation, attributes, and data. Every section starts with "@".

ARFF files take Nominal, Numeric, String, Date, and Relational data attributes. Some of the well-known machine learning datasets is present in WEKA as ARFF.

Format for ARFF is:

```
@relation <relation name>
@attribute <attribute name and data type>
@data
```

An example of an ARFF file is:

```
@relation weather
@attribute outlook {sunny, overcast, rainy}:
@attribute temperature real
@attribute humidity real
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}    //class attribute: It represents the output.

@data
sunny, FALSE,85,85,no
sunny, TRUE,80,90,no
overcast, FALSE,83,86,yes
rainy, FALSE,70,96,yes
rainy, FALSE,68,80,yes
```

CSV - Comma Separated Values

CSV is a very old, very simple and very common "standard" for (tabular) data. We say "standard" in quotes because there was never a formal standard for CSV, though in 2005 someone did put together a RFC for it.

CSV is supported by a **huge** number of tools from spreadsheets like Excel, Open Office and Google Docs to complex databases to almost all programming languages. As such it is probably the most widely supported structured data format in the world.

CSV Format

- CSV is probably the simplest possible structured format for data
- CSV strikes a delicate balance, remaining readable by both machines and humans
- CSV is a two dimensional structure consisting of rows of data, each row containing multiple cells. Rows are (usually) separated by line terminators so each row corresponds to one line. Cells within a row are separated by commas.

- Note that strictly we're really talking about DSV files in that we can allow 'delimiters' between cells other than a comma. However, many people and many programs still call such data CSV (since comma is so common as the delimiter)
- CSV is a "text-based" format, i.e. a CSV file *is* a text file. This makes it amenable for processing with all kinds of text-oriented tools (from text editors to unix tools like sed etc..)

What a CSV looks like?

If you open up a CSV file in a text editor it would look something like:

A,B,C
1,2,3
4,"5,3",6

Here there are 3 rows each of 3 columns. Notice how the second column in the last line is "quoted" because the content of that value actually contains a "," character.

Without the quotes this character would be interpreted as a column separator. To avoid this confusion we put quotes around the whole value. The result is that we have 3 rows each of 3 columns (Note a CSV file does not *have* to have the same number of columns in each row).

2.6.3 XRFF Data Format

XRFF stands for the XML attribute Relation File Format. It represents data that can store comments, attributes, and instance weights. It has .xrff extension and .xrff.gz (compressed format) file extension. The XRFF files represented data in XML format.

2.6.4 Database Connectivity

With WEKA, it is easy to connect to a database using a JDBC driver. JDBC driver is necessary to connect to the database, for example:
MS SQL Server (com.microsoft.jdbc.sqlserver.SQLServerDriver)
Oracle (oracle.jdbc.driver.OracleDriver)

2.6.5 Classifiers

To predict the output data, WEKA contains classifiers. The classification algorithms available for learning are decision-trees, support vector machines, instance-based classifiers, and logistic regression, and Bayesian networks. Depending upon the requirement using trial and test, the user can find out a suitable algorithm for the analysis of data. Classifiers are used to classify the data sets based on the characteristics of the attributes.

2.6.6 Clustering

WEKA uses the Cluster tab to predict the similarities in the dataset. Based on clustering, the user can find out the attributes useful for analysis and ignore

other attributes. The available algorithms for clustering in WEKA are k-means, EM, Cobweb, X-means, and FarhtestFirst.

2.6.7 Association

The only algorithm available in WEKA for finding out association rules is ***Apriori.***

2.6.8 Attribute Section Measures

WEKA uses 2 approaches for best attribute selection for calculation purpose:

- **Using Search method algorithm:** Best-first, forward selection, random, exhaustive, genetic algorithm, and ranking algorithm.
- **Using Evaluation method algorithms:** Correlation-based, wrapper, information gain, chi-squared.

2.6.9 Visualization

WEKA supports the 2D representation of data, 3D visualizations with rotation, and 1D representation of single attribute. It has the "Jitter" option for nominal attributes and "hidden" data points.

Example 1: To create a data file in *.arff* format manually using any editor such as notepad or turbo editor.

An ***arff*** (attribute-relation file format) file is an ASCII text file that describes a list of instances sharing a set of attributes. These files have two distinct sections, **Header** information and **Data** information. The **Header** of the ***arff*** file contains the name of the relation, a list of the attributes (the columns in the data), and their types. The ***arff*** Header section of the file contains the relation declaration and attributes declarations.

The @relation Declaration:-The relation name is defined as the first line in the ARFF file, The format is: @relation <relation-name>, where <relation-name> is a string, The string must be quoted if the name includes spaces.
Examples:

@relation iris

@relation bank

The @attribute Declarations:-Attribute declarations take the form of an ordered sequence of **@attribute** statements. Each attribute in the data set has its own **@attribute** statement which uniquely defines the name of that attribute and its data type. The order the attributes are declared indicates the column position in the data section of the file. For example, if an attribute is the third one declared then Weka expects that all that attributes values will be found in the third comma delimited column. The format for the **@attribute** statement is: @attribute <attribute-name> <data type>, where the <*attribute-name*> must start with an alphabetic character. If spaces are to be included in the name then the entire name must be quoted. The <*datatype*> can be any of

the four types currently (version 3.2.1) supported by Weka. Numeric, <nominal-specification>, string, date [<date-format>].

Numeric attributes can be real or integer numbers. Nominal attributes:- Nominal values are defined by providing an <nominal-specification> listing the possible values: {<nominal-name1>, <nominal-name2>, <nominal-name3>, ...} For example, @ATTRIBUTE gender {male, female}. String attributes allow us to create attributes containing arbitrary textual values. Date attribute declarations take the form: @attribute <name> date [<date-format>] where <name> is the name for the attribute. <date-format> is an optional string specifying how date values should be parsed and printed (this is the same format used by SimpleDateFormat). The default format string accepts the ISO-8601 combined date and time format: "yyyy-MM-dd'T'HH:mm:ss".

Examples
@attribute petalwidth NUMERIC

@ attribute class {First, Second, Third}

@ attribute name STRING

@ attribute dob DATE

@ attribute doj DATE "yyyy-MM-dd HH:mm:ss".

ARFF Data Section:-The ARFF Data section of the file contains the data declaration line and the actual instance lines. The **@data** declaration is a single line denoting the start of the data segment in the file. The format is:@data . Each instance is represented on a single line, with carriage returns denoting the end of the instance. Attribute values for each instance are delimited by commas. They must appear in the order that they were declared in the header section (i.e. the data corresponding to the nth **@attribute** declaration is always the nth field of the attribute).

Examples
@data 5.1,3.5,1.4,0.2,Iris-setosa

Example

@relation bank_data

@attribute name string
@attribute sex {FEMALE,MALE}
@attribute region {INNER_CITY,TOWN,RURAL,SUBURBAN}
@attribute income numeric
@attribute married {NO,YES}
@attribute car {NO,YES}

@data
Xyz,FEMALE,INNER_CITY,17546,NO,YES
Abc,MALE,RURAL,100000,YES,YES

Example 2: Create an Employee dataset using any editor such as notepad or turbo editor.

Description: We need to create an Employee Table with training data set which includes attributes like name, id, salary, experience, gender, phone number.

Step by step procedure:

1) Open Notepad
2) Type the following training data set with the help of Notepad for Employee dataset.

```
@relation employee
@attribute name {x,y,z,a,b}
@attribute id numeric
@attribute salary {low,medium,high}
@attribute exp numeric
@attribute gender {male,female}
@attribute phone numeric

@data
x,101,low,2,male,250311
y,102,high,3,female,251665
z,103,medium,1,male,240238
a,104,low,5,female,200200
b,105,high,2,male,240240
```

3) After that the file is saved with ***.arff*** file format.
4) Minimize the ***.arff*** file and then open WEKA
5) Click on WEKA, then Weka dialog box is displayed on the screen.
6) In that dialog box there are four modes, click on explorer.
7) Explorer shows many options. click on 'open file' and select the ***arff*** file
8) Click on edit button which shows employee table on WEKA.

Example 3: Convert the EXCEL dataset into *arff* file

Step to step procedure:

1) Convert the Excel data set into CSV (comma separated value) format.
2) One easy way to do this is to load it into Excel and use "Save As" to save the file in CSV format.
3) Edit the CSV file, and add the ***arff*** header information to the file.
4) This involves creating the @relation line, one @attribute line per attribute and @data to signify the start of data.
5) Finally save this file with (.arff) extension.
6) It is also considered good practice to add comments at the top of the file describing where you obtained this data set, what its summary characteristics are, etc.
7) A comment in the ***arff*** format is started with the percent character % and continues until the end of the line.
8) Open this file with WEKA tool.

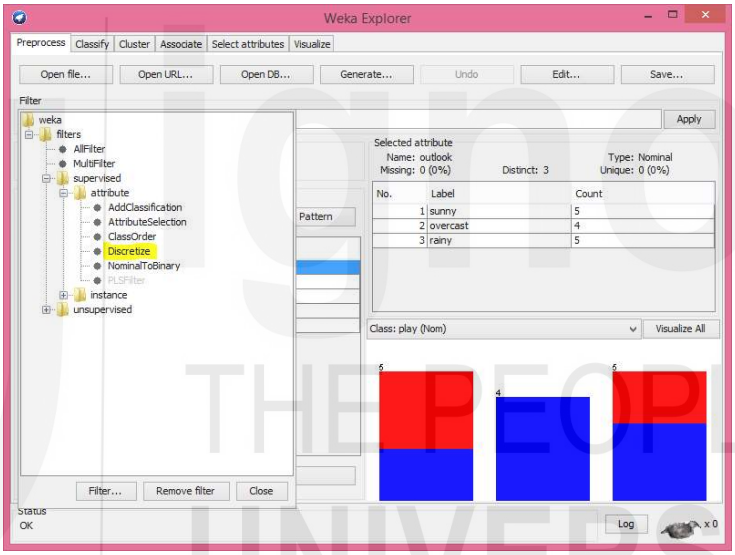
2.7 DATA PREPROCESSING

2.7.1 Discretization

Dicretization of numerical data is one of the most influential data preprocessing tasks in knowledge discover and data mining.

Discretization Process – In supervised learning and specifically in classification discredit the data in the columns to enable the use of the algorithms to produce a mining model. Discretization is the process of putting values into buckets so that there are a limited number of possible states. The buckets themselves are treated as ordered and discrete values.

- Start→ weka→ select the explorer→select the dataset (weather.nominal).
- Choose discretize filter.



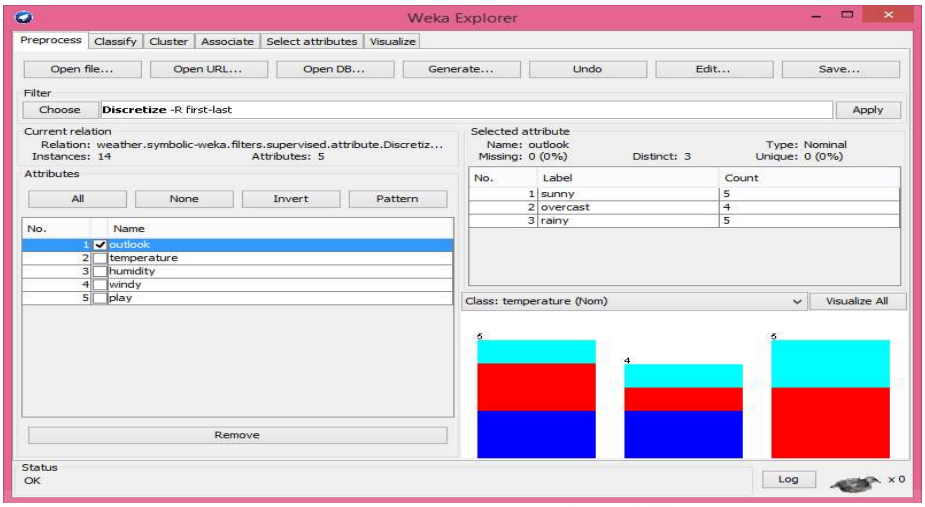
Synopsis

An instance filter that discreteness a range of numeric attributes in the dataset into nominal attributes. Discretization is by Fayyad & Irani's MDL method (the default).

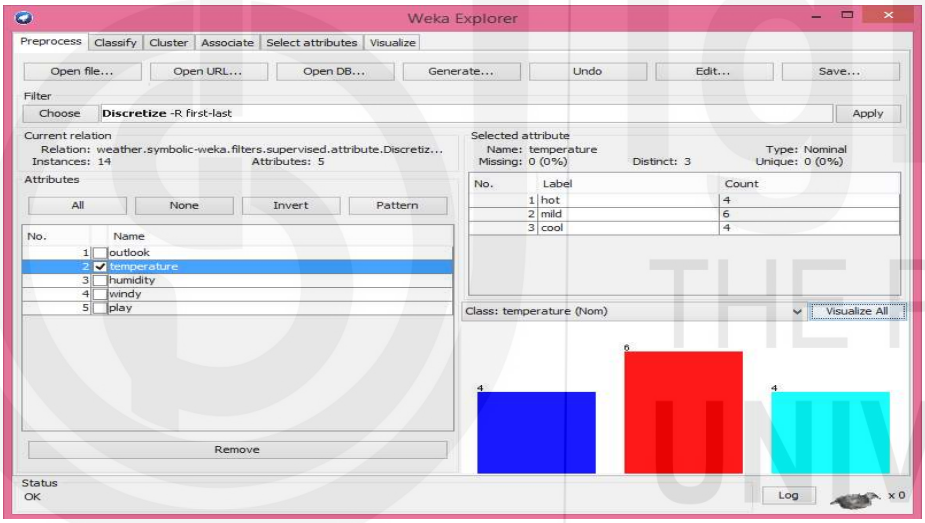
Options

attributeIndices -- Specify range of attributes to act on. This is a comma separated list of attribute indices, with "first" and "last" valid values. Specify an inclusive range with "-". Example: "first-3,5,6-10,last".
invertSelection -- Set attribute selection mode. If false, only selected (numeric) attributes in the range will be discretized; if true, only non-selected attributes will be discretized.
makeBinary -- Make resulting attributes binary.
useBetterEncoding -- Uses a more efficient split point encoding.
useKononenko -- Use Kononenko's MDL criterion. If set to false uses the Fayyad & Irani criterion.

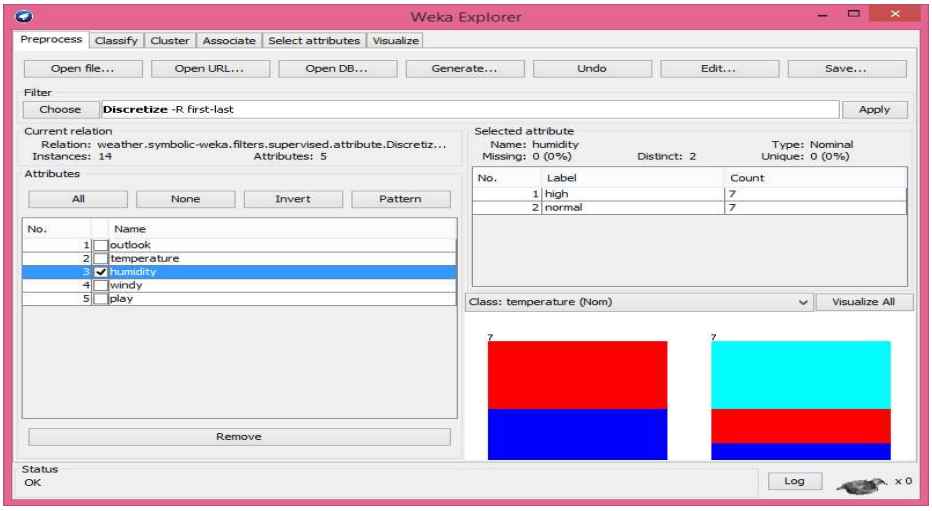
Select the outlook attribute based on class temperature to visualize below :



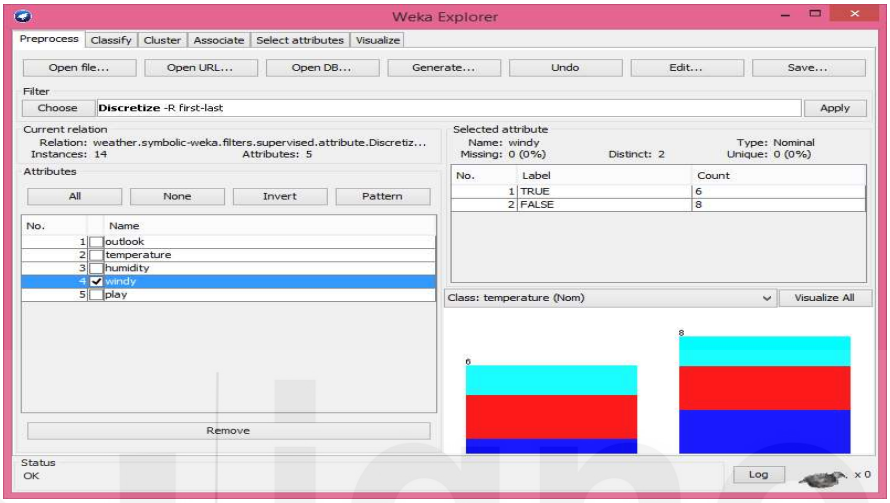
Select the temperature attribute based on class temperature to visualize below:



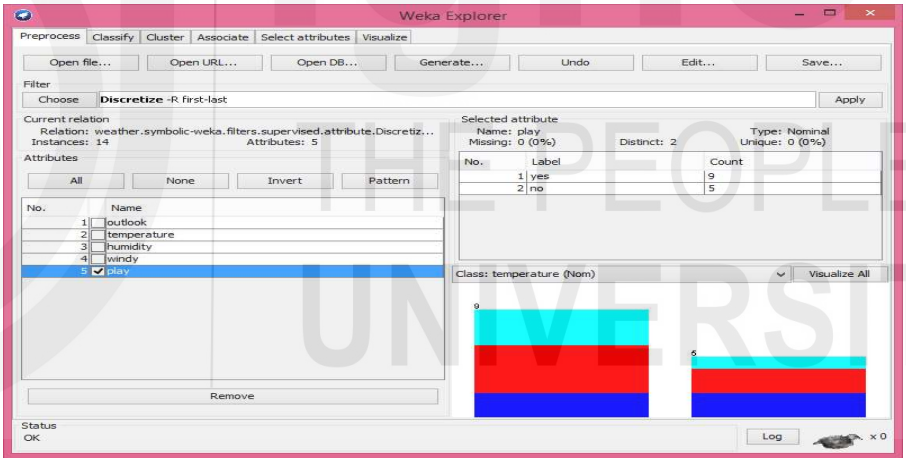
Select the humidity attribute based on class temperature to visualize below:



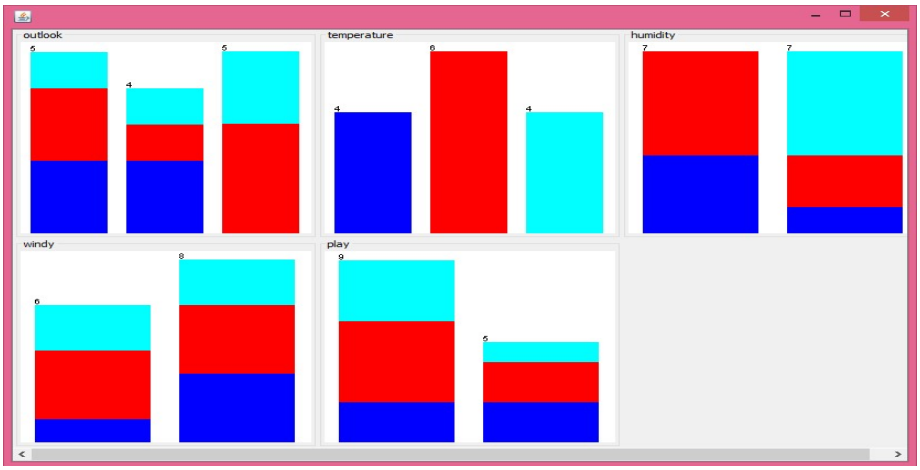
Select the windy attribute based on class temperature to visualize below:



Select the play attribute based on class temperature to visualize below:



Comparison of all visualize results:

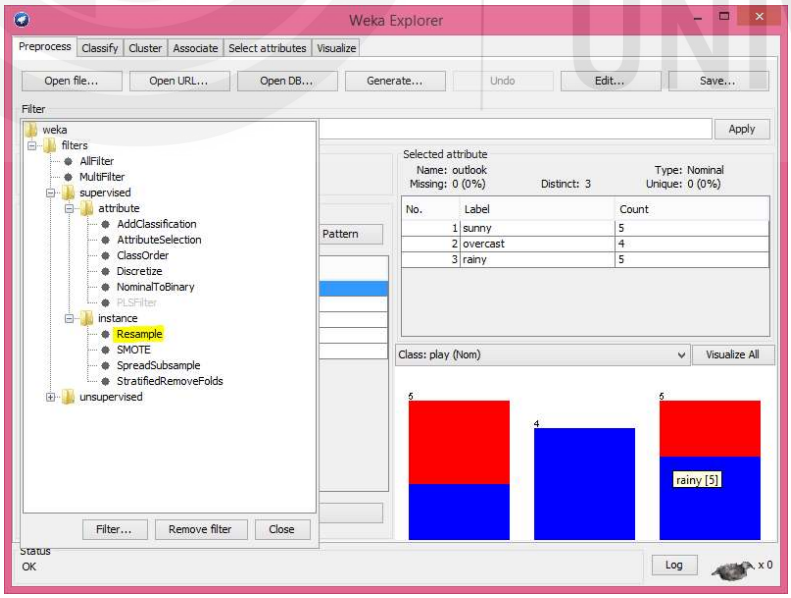


2.7.2 Random Sampling

Filters instances according to the value of an attribute produces a randomsubsample of a dataset using either sampling with replacement or without replacement. Produces a random subsample of a dataset using the reservoirsampling Algorithm "R" by Vitter.

The original dataset must fit entirely in memory. The number of instances in the generated dataset may be specified. The dataset must have a nominal class attribute. If not, use the unsupervised version. The filter can be made to maintain the class distribution in the subsample, or to bias the class distribution toward a uniform distribution. When used in batch mode subsequent batches are NOT re-sampled.

Start→ weka→ select the explorer→select the dataset (weather.nominal).
Choose resample filter.



Synopsis

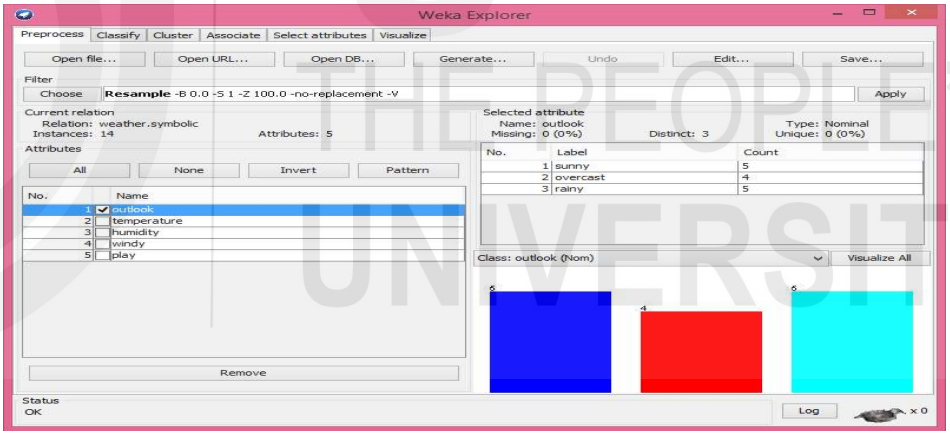
Produces a random subsample of a dataset using either sampling with replacement or without replacement. The original dataset must fit entirely in memory. The number of instances in the generated dataset may be specified. The dataset must have a nominal class attribute. If not, use the unsupervised version. The filter can be made to maintain the class distribution in the subsample, or to bias the class distribution toward a uniform distribution. When used in batch mode (i.e. in the Filtered Classifier), subsequent batches are NOT resampled.

Options

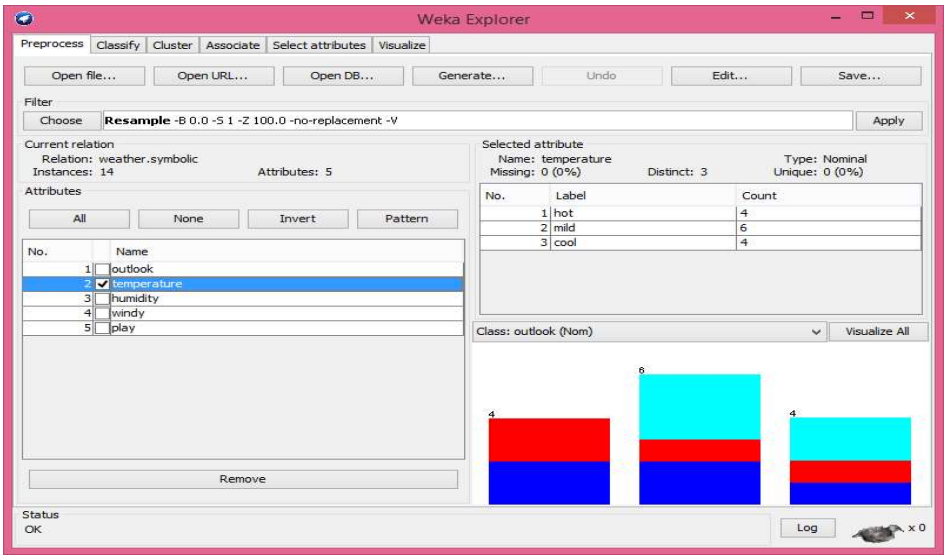
randomSeed -- Sets the random number seed for subsampling.
biasToUniformClass -- Whether to use bias towards a uniform class. A value of 0 leaves the class distribution as-is, a value of 1 ensures the class distribution is uniform in the output data.

debug -- If set to true, filter may output additional info to the console.
noReplacement -- Disables the replacement of instances.
doNotCheckCapabilities -- If set, filters capabilities are not checked before filter is built (Use with caution to reduce runtime).
sampleSizePercent -- The subsample size as a percentage of the original set.
invertSelection -- Inverts the selection (only if instances are drawn WITHOUT replacement).

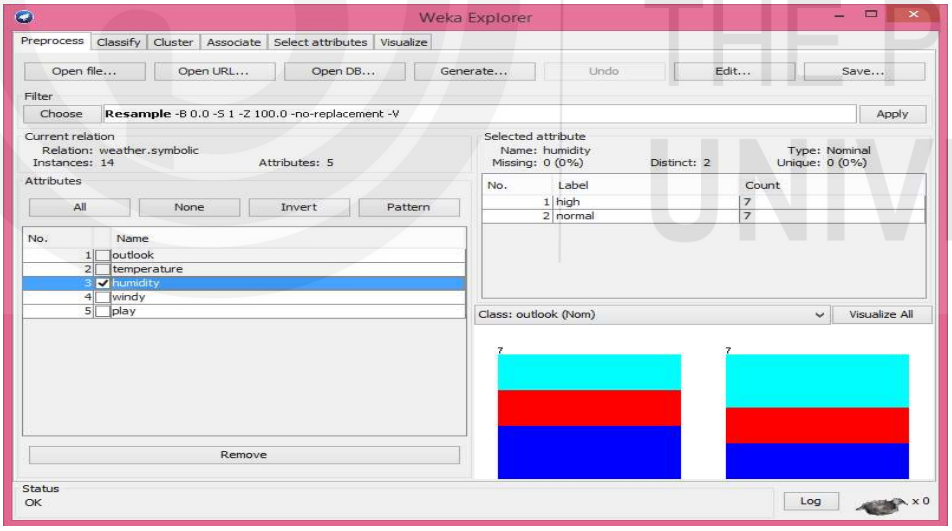
Select the outlook attribute based on class outlook (Nom) to visualize below:



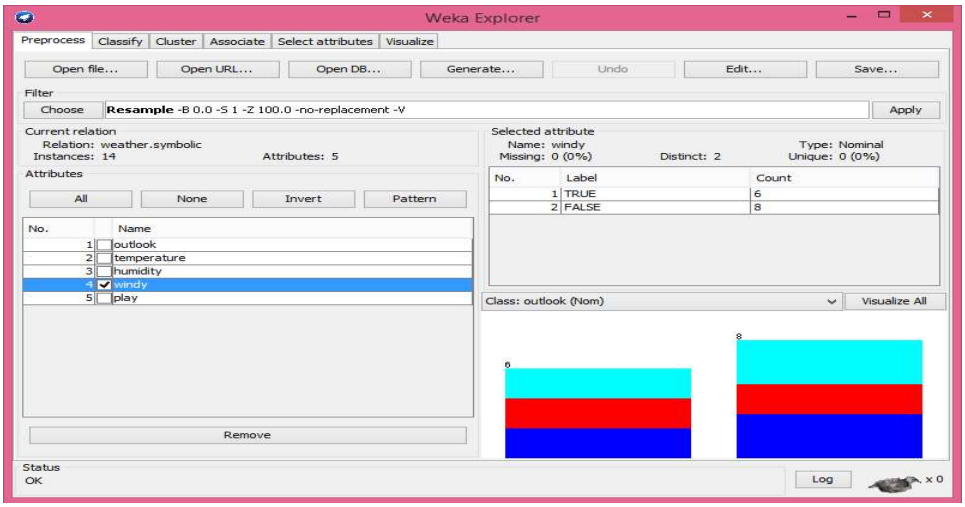
Select the temperature attribute based on class outlook (Nom) to visualize below:



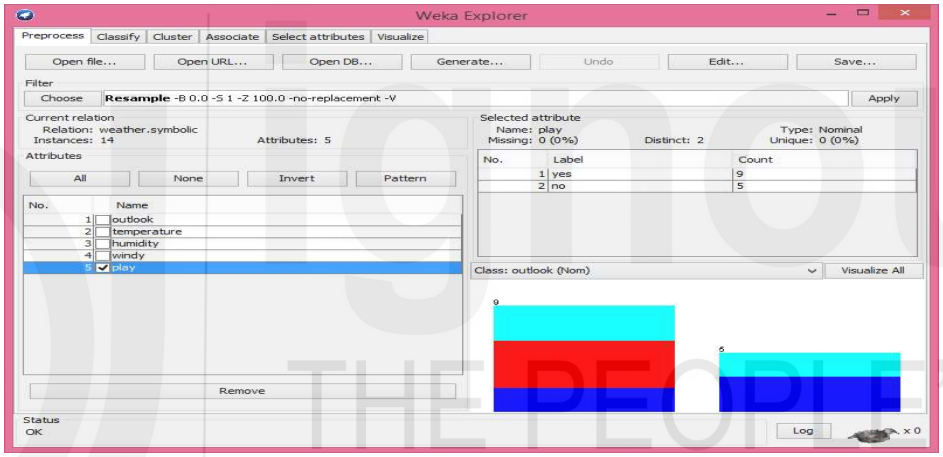
Select the humidity attribute based on class outlook (Nom) to visualize below:



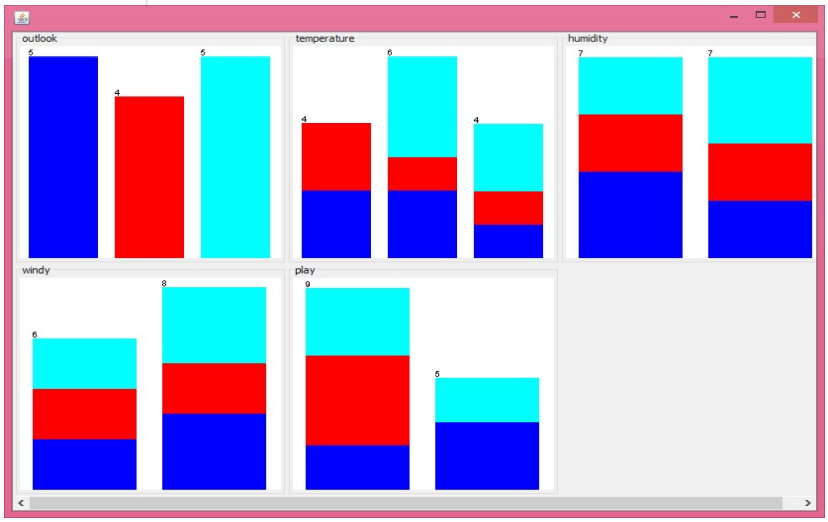
Select the windy attribute based on class outlook (Nom) to visualize below:



Select the play attribute based on class outlook (Nom) to visualize below:



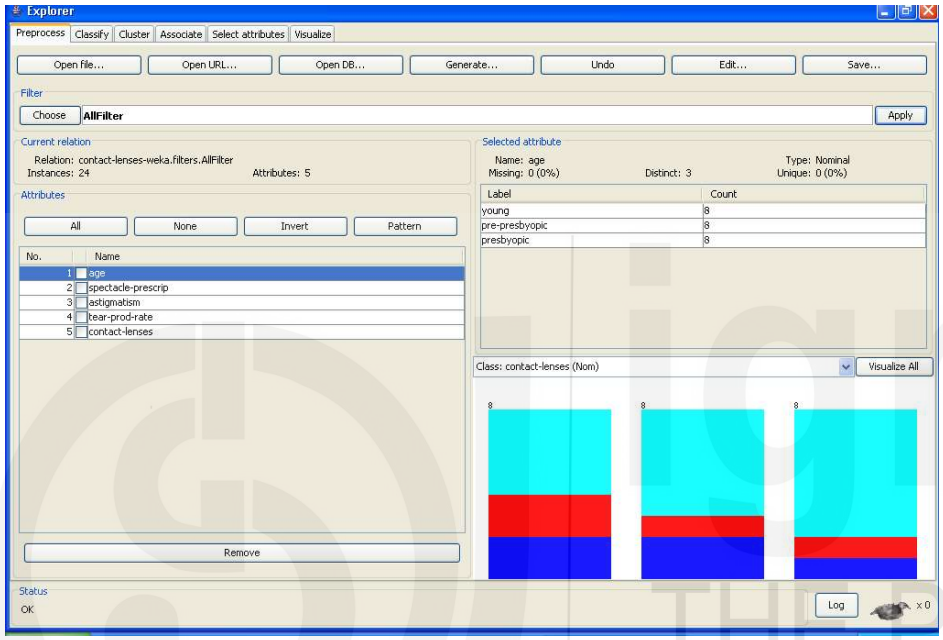
Comparison of all visualize results:



Example: Demonstrate preprocessing of the data in WEKA.

Step by step procedure:

1) Click on openfile button and select an arff file to choose dataset for analysis.
2) Apply filtering options to select required dataset.
3) We can select and remove attributes which are irrelevant to the analysis.
4) We can visualize the attribute vise classification from the visualization frame.
5) The output window appears as follows:

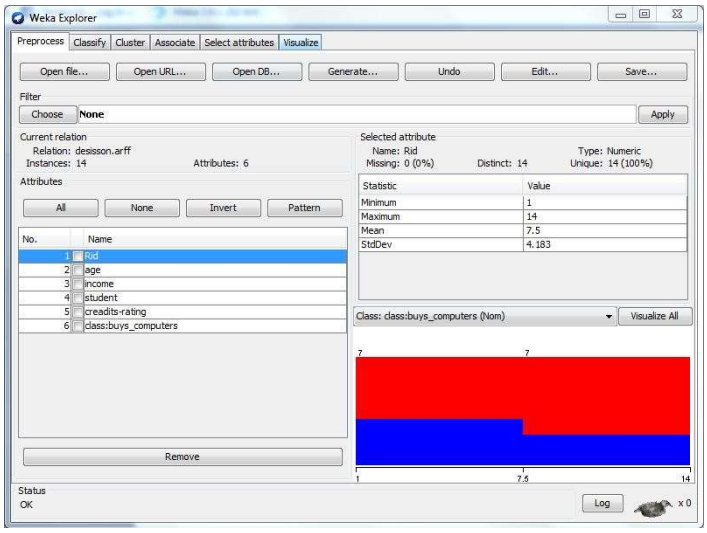


2.8 ATTRIBUTE SELECTION IN WEKA

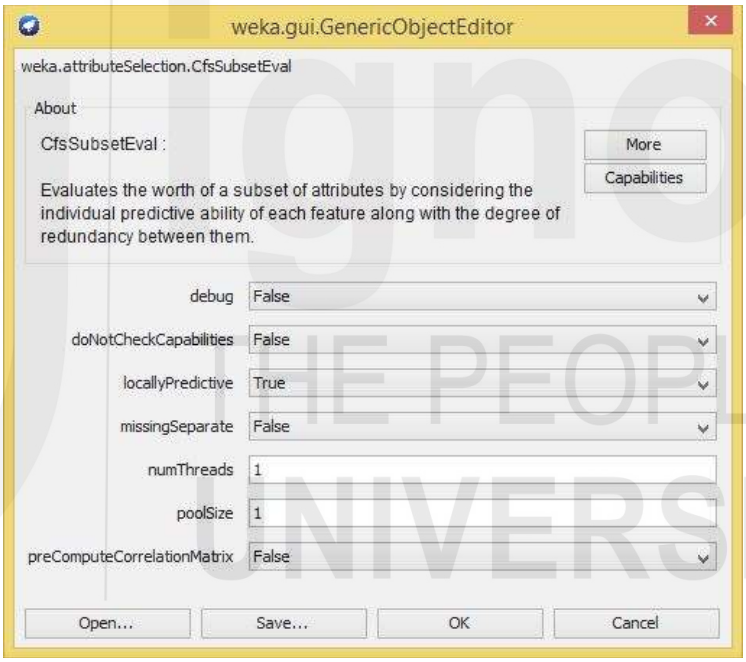
Attribute selection or variable subset selection is the process of selecting a subset of relevant features in this there are four algorithms are as follows:

- CFS Subset evaluated algorithm
- Information gain algorithm
- Correlation attribute evaluated algorithm
- Gain ratio attribute evaluated algorithm

Start-->weka-->select Explorer-->select dataset (Buys Computers (if not available create one))



2.8.1 CFS Subset Evaluated Algorithm



It evaluates the worth of a subset of attributes by considering the individual predictive ability of each feature along with the degree of redundancy between them.

Subsets of features that are highly correlated with the class while having low inter-correlation are preferred.

Options

numThreads: The number of threads to use, which should be >= size of thread pool.

Debug: Output debugging info

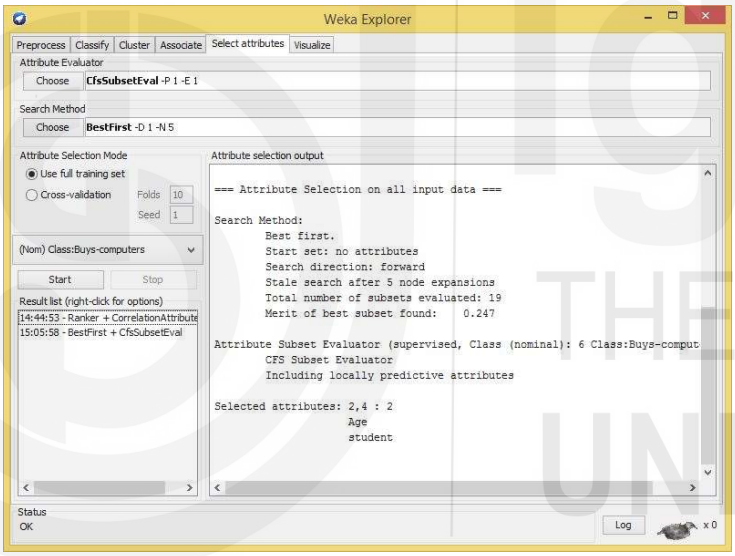
MissingSeparate: Treat missing as a separate value. Otherwise, counts for missing values are distributed across other values in proportion to their frequency.

PoolSize: The size of the thread pool, for example, the number of cores in the CPU.

DoNotCheckCapabilities: If set, evaluator capabilities are not checked before evaluator is built (Use with caution to reduce runtime).

PreComputeCorrelationMatrix: Recomputed the full correlation matrix at the outset, rather than compute correlations lazily (as needed) during the search. Use this in conjunction with parallel processing in order to speed up a backward search.

LocallyPredictive: Identify locally predictive attributes. Iteratively adds attributes with the highest correlation with the class as long as there is not already an attribute in the subset that has a higher correlation with the attribute in question weka.attributeSelection.CfsSubsetEval



Attribute selection CfsSubsetEval Output

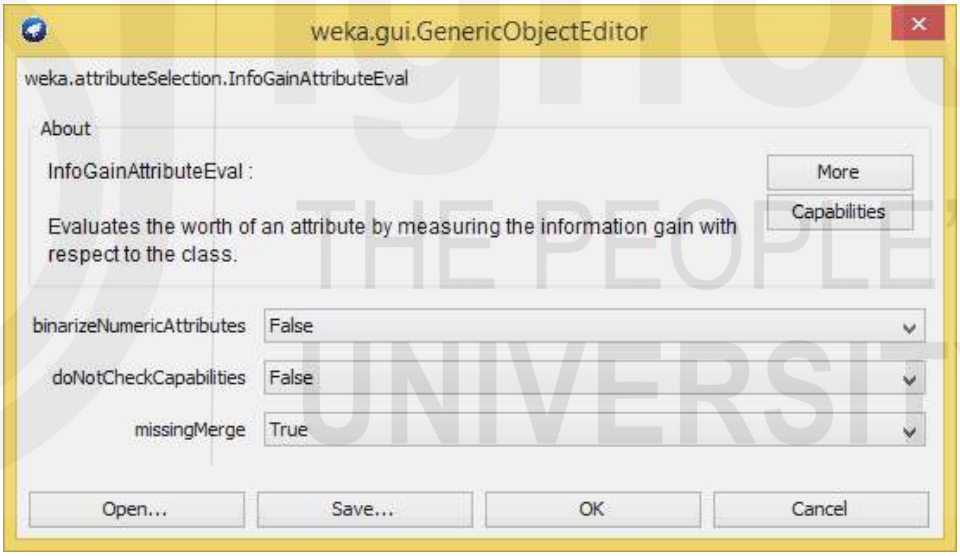
=== Run information ===
Evaluator: weka.attributeSelection.CfsSubsetEval -P 1 -E 1
Search: weka.attributeSelection.BestFirst -D 1 -N 5
Relation: dessision.arff
Instances: 14
Attributes: 6
Rid
Age
Income
student
Credit-Rating
Class: Buys-computers

Evaluation mode: evaluate on all training data

=== Attribute Selection on all input data ===
Search Method:
Best first.
Start set: no attributes
Search direction: forward
Stale search after 5 node expansions
Total number of subsets evaluated: 19
Merit of best subset found: 0.247
Attribute Subset Evaluator (supervised, Class (nominal): 6 Class: Buys-computers):
CFS Subset Evaluator
Including locally predictive attributes
Selected attributes: 2, 4 : 2
Age
student

2.8.2 Information Gain Algorithm

weka.attributeSelection.information gain algorithm

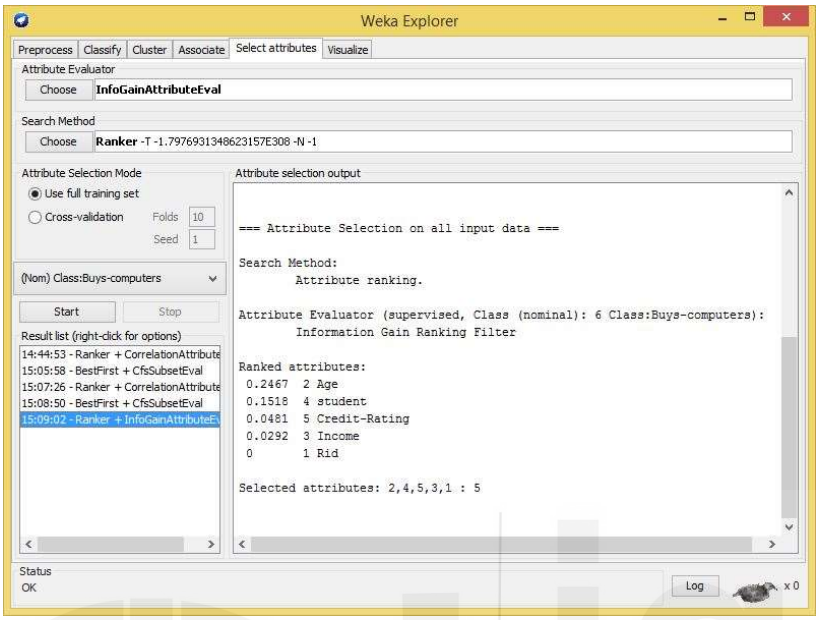


Algorithm: Information Gain Algorithm

InfoGainAttributeEval:
Evaluates the worth of an attribute by measuring the information gain with respect to the class.
InfoGain (Class, Attribute) = H (Class) - H (Class | Attribute).

Options
missingMerge -- Distribute counts for missing values. Counts are distributed across other values in proportion to their frequency. Otherwise, missing is treated as a separate value.
binarizeNumericAttributes -- Just binaries numeric attributes instead of properly discrediting them.

doNotCheckCapabilities -- If set, evaluator capabilities are not checked before evaluator is built (Use with caution to reduce runtime).



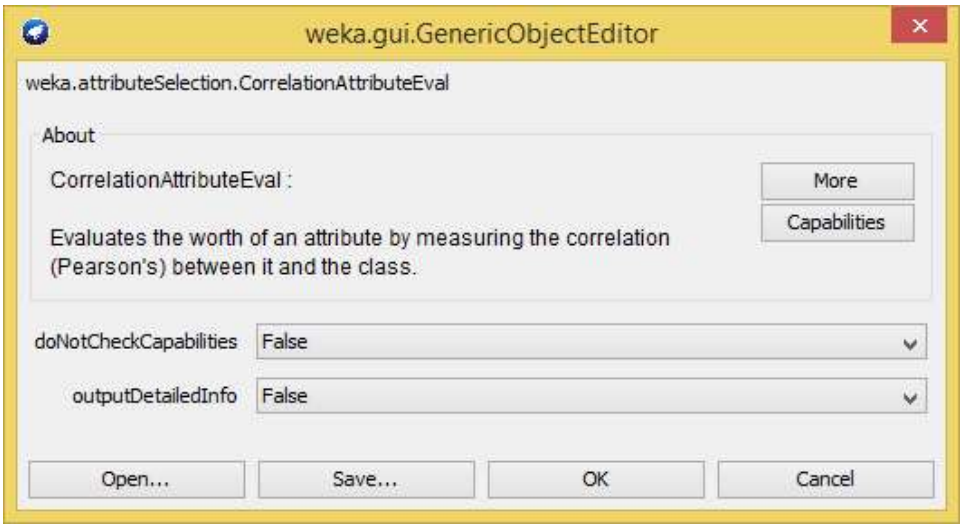
Attribute selection output

=== Run information ===
Evaluator: weka.attributeSelection.InfoGainAttributeEval
Search: weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1
Relation: dessision.arff
Instances: 14
Attributes: 6
Evaluation mode: evaluate on all training data

=== Attribute Selection on all input data ===
Search Method:
Attribute ranking.
Attribute Evaluator (supervised, Class (nominal): 6 Class: Buys-computers):
Information Gain Ranking Filter

Ranked attributes:
0.2467 2 Age
0.1518 4 student
0.0481 5 Credit-Rating
0.0292 3 Income
0 1 Rid
Selected attributes: 2, 4, 5, 3, 1 : 5

2.8.3 Correlation Attribute Evaluated Algorithm



weka.attributeSelection.CorrelationAttributeEval

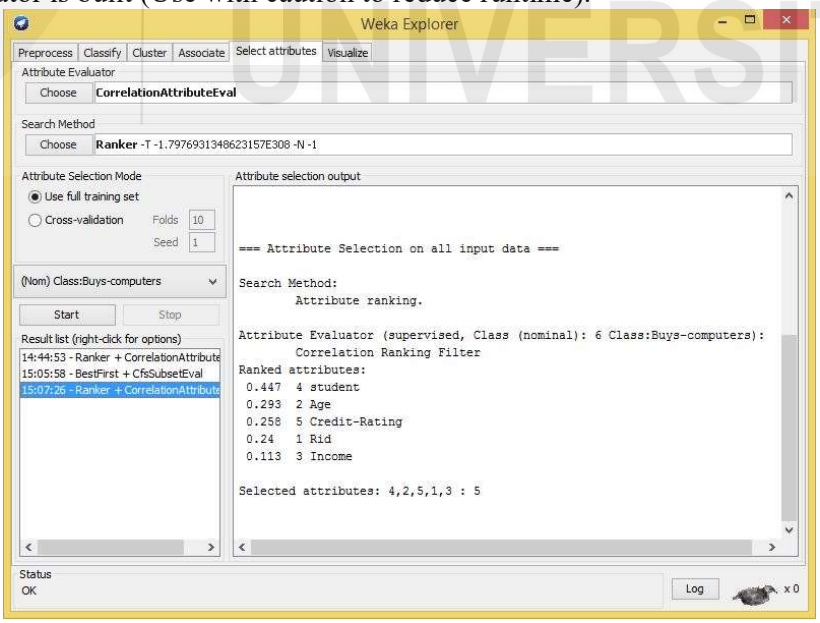
CorrelationAttributeEval

Evaluates the worth of an attribute by measuring the correlation (Pearson's) between it and the class.
Nominal attributes are considered on a value by value basis by treating each value as an indicator. An overall correlation for a nominal attribute is arrived at via a weighted average.

Options

OutputDetailedInfo Output per value correlation for nominal attributes

DoNotCheckCapabilities If set, evaluator capabilities are not checked before evaluator is built (Use with caution to reduce runtime).



Attribute selection correlation attribute eval

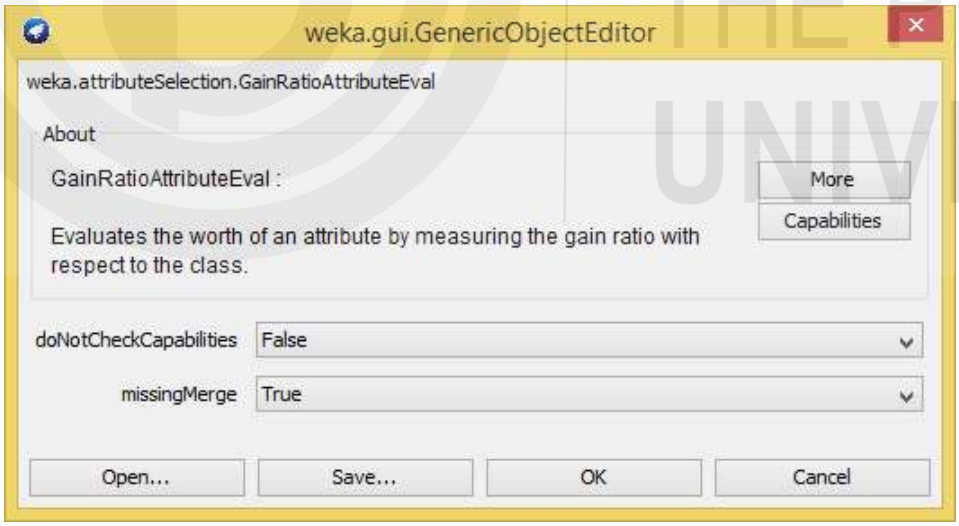
Output:

```
=== Run information ===
Evaluator:    weka.attributeSelection.CorrelationAttributeEval
Search:       weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N
-1
Relation:     dessision.arff
Instances:    14
Attributes:   6
Evaluation mode:
evaluate on all training data
=== Attribute Selection on all input data ===
Search Method:
Attribute ranking.
Attribute Evaluator (supervised, Class (nominal): 6 Class: Buys-computers):
        Correlation Ranking Filter

Ranked attributes:
 0.447  4 student
 0.293  2 Age
 0.258  5 Credit-Rating
 0.24   1 Rid
 0.113  3 Income
Selected attributes: 4, 2, 5, 1, 3  : 5
```

2.8.4 Gain Ratio Attribute Evaluated Algorithm:

weka.attributeSelection.GainRatioAttributeEval



GainRatioAttributeEval

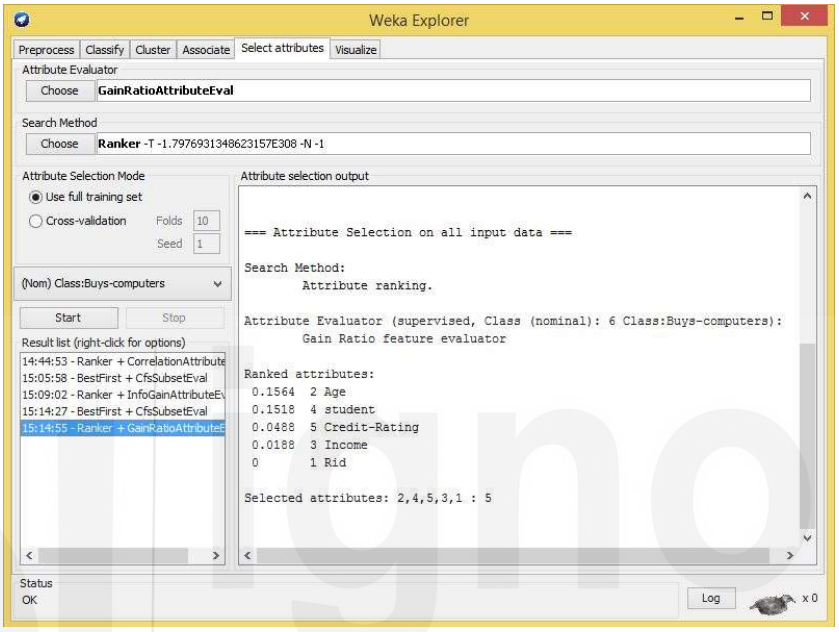
Evaluates the worth of an attribute by measuring the gain ratio with respect to the class.

GainR(Class, Attribute) = (H(Class) - H(Class | Attribute)) / H(Attribute).

Options

MissingMerge: Distribute counts for missing values. Counts are distributed across other values in proportion to their frequency. Otherwise, missing is treated as a separate value.

DoNotCheckCapabilities: If set, evaluator capabilities are not checked before evaluator is built (Use with caution to reduce runtime).



Attribute selection gain ratio attribute algorithm output

=== Run information ===
Evaluator: weka.attributeSelection.GainRatioAttributeEval
Search: weka.attributeSelection.Ranker -T -1.7976931348623157E308 -N -1
Relation: dessision.arff
Instances: 14
Attributes: 6

Evaluation mode: evaluate on all training data

=== Attribute Selection on all input data ==
Search Method:
Attribute ranking.
Attribute Evaluator (supervised, Class (nominal): 6 Class: Buys-computers):
Gain Ratio feature evaluator

Ranked attributes:
0.1564 2 Age
0.1518 4student
0.0488 5 Credit-Rating
0.0188 3 Income

0 1 Rid

Selected attributes: 2, 4, 5, 3, 1 : 5

2.9 ASSOCIATION RULE MINING

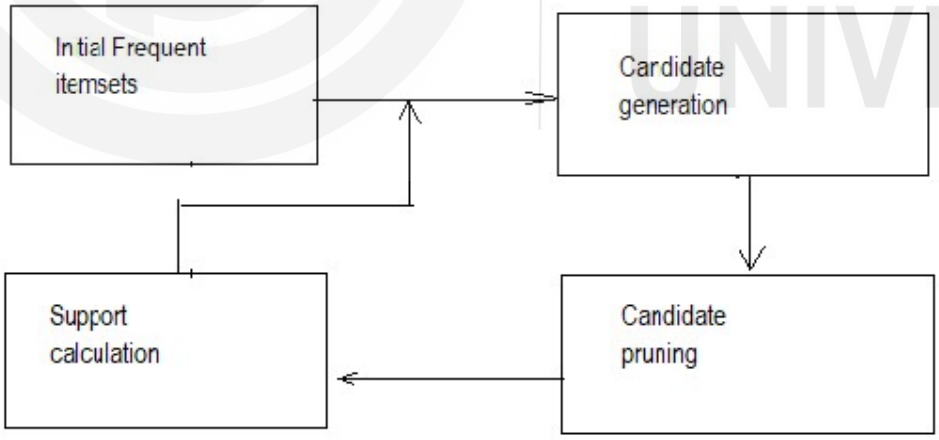
The Association rule mining is a procedure which is meant to find frequent patterns, correlations, associations, or causal structures from data sets found in various kinds of databases such as relational databases, transactional databases, and other forms of data repositories.

2.9.1 Apriori Algorithm

Apriori is an algorithm for frequent item set mining and association rule learning over transitional databases. It proceeds by identifying the frequent individual items in the database and attending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. The frequent item sets determine by apriori can be used to define by association rules which highlight tends of the databases.

The Apriori algorithm was proposed by Agrawal and Srikant in 1994. Apriori is designed to operate on database containing transactions. Each transaction is seen as a set of items.Given a threshold, the Apriori algorithm identifies the item sets which are subsets of atleast transactions in the database

Apriori uses breath-first search and a Hash tree structure to count candidate item sets efficiently. It generates candidate item sets of length k from item sets of length k-1. Then it prunes the candidates which have an infrequent sub pattern.



Pseudo code:

```
Apriori(T, ε)
    L_1 ← {large 1 − itemsets}
    k ← 2
    while L_{k−1} ≠ ∅
        C_k ← {a ∪ {b} | a ∈ L_{k−1} ∧ b ∉ a} − {c | {s | s ⊆ c ∧ |s| = k − 1} ⊈ L_{k−1}}
        for transactions t ∈ T
            C_t ← {c | c ∈ C_k ∧ c ⊆ t}
            for candidates c ∈ C_t
                count[c] ← count[c] + 1
        L_k ← {c | c ∈ C_k ∧ count[c] ≥ ε}
        k ← k + 1
    return ⋃_k L_k
```

In metric type there are four metrics:

- Confidence
- Lift
- Leverage
- Conviction

Useful Concepts

To select interesting rules from the set of all possible rules, constraints on various measures of significance and interest can be used. The best-known constraints are minimum thresholds on support and confidence.

(i) Support

The support supp(X) of an item set X is defined as the proportion of transactions in the data set which contain the item set.
Supp(X) =no of transactions which contain the item set X/total no of transactions

(ii) Confidence

The confidence of a rule is defined
Conf(x->y) = supp (XUY)/supp(x)

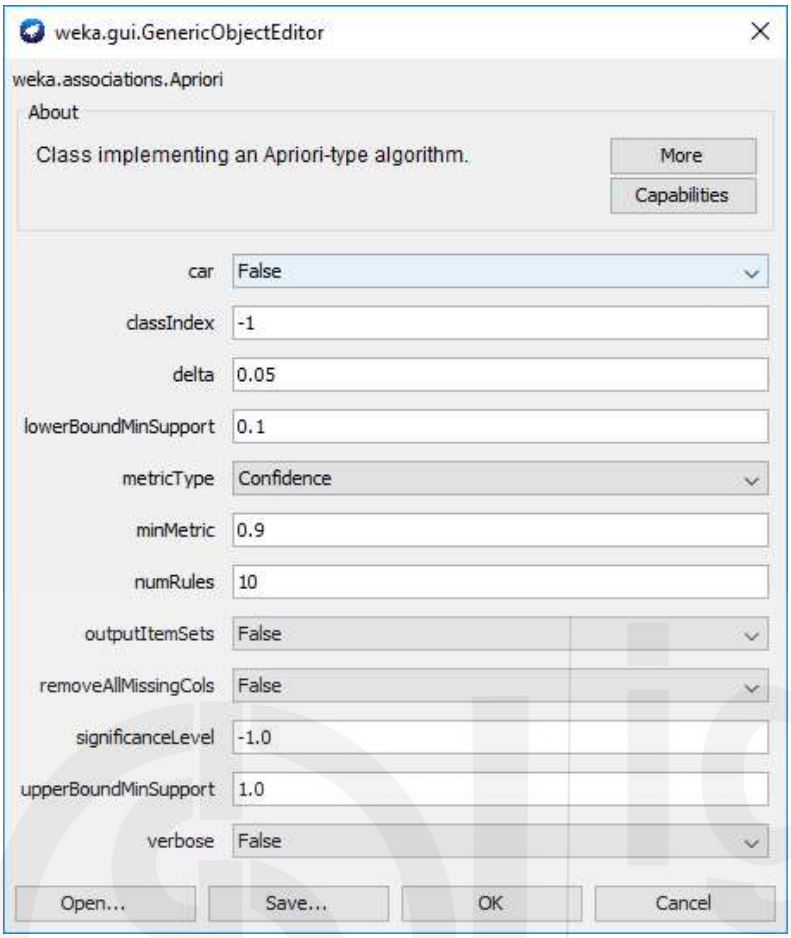
(iii) Lift

The lift of a rule is defined as:
Lift(X->Y) = supp (XUY)/supp(Y)*supp(X)

(iv) Conviction

The conviction of a rule is defined as:
Conv (X_>Y) =1-supp(Y)/1-conf(X->Y)

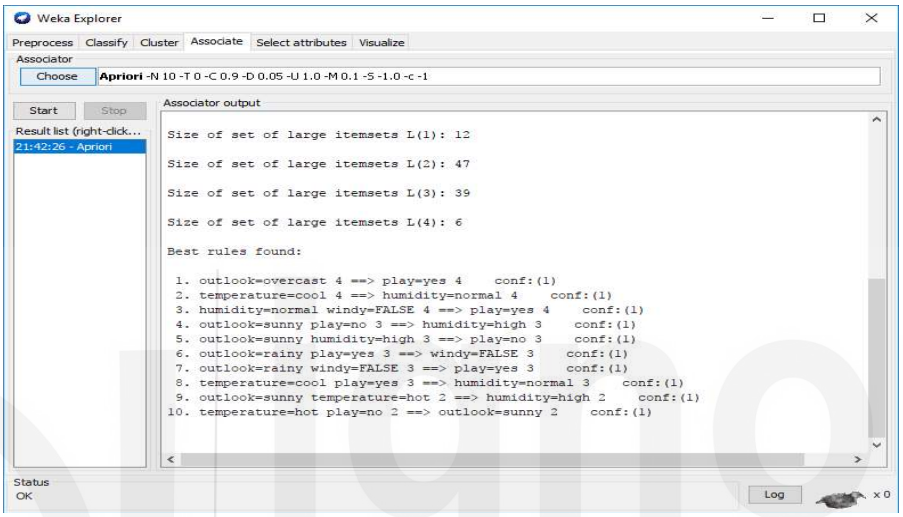
Options



car: If enabled class association rules are mined instead of (general) association rules.
classIndex: Index of the class attribute. If set to -1, the last attribute is taken as class attribute.
delta: Iteratively decrease support by this factor. Reduces support until min support is reached or required number of rules has been generated.
LowerBoundMinSupport: Lower bound for minimum support.
MetricType: Set the type of metric by which to rank rules. Confidence is the proportion of the examples covered by the premise that are also covered by the consequence (Class association rules can only be mined using confidence). Lift is confidence divided by the proportion of all examples that are covered by the consequence. This is a measure of the importance of the association that is independent of support. Leverage is the proportion of additional examples covered by both the premise and consequence above those expected if the premise and consequence were independent of each other. The total number of examples that this represents is presented in brackets following the leverage. Conviction is another measure of departure from independence.
MinMetric : Minimum metric score. Consider only rules with scores higher than this value.
NumRules : Number of rules to find.
OutputItemSets : If enabled the item sets are output as well.
RemoveAllMissingCols : Remove columns with all missing values.
SignificanceLevel: Significance level. Significance test (confidence metric only).

UpperBoundMinSupport: Upper bound for minimum support. Start iteratively decreasing minimum support from this value.
Verbose : If enabled the algorithm will be run in verbose mode.
How to open the Apriori Algorithm in WEKA
Start→weka→select the explorer→select the open file(weather Nominal)→select the Association →chooses the algorithm(Apriori)→click on start.

(a) Using 10 numrules by using confidence



=== Run information ===
Scheme: weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation: weather.symbolic
Instances: 14
Attributes: 5
outlook
temperature
humidity
windy
play

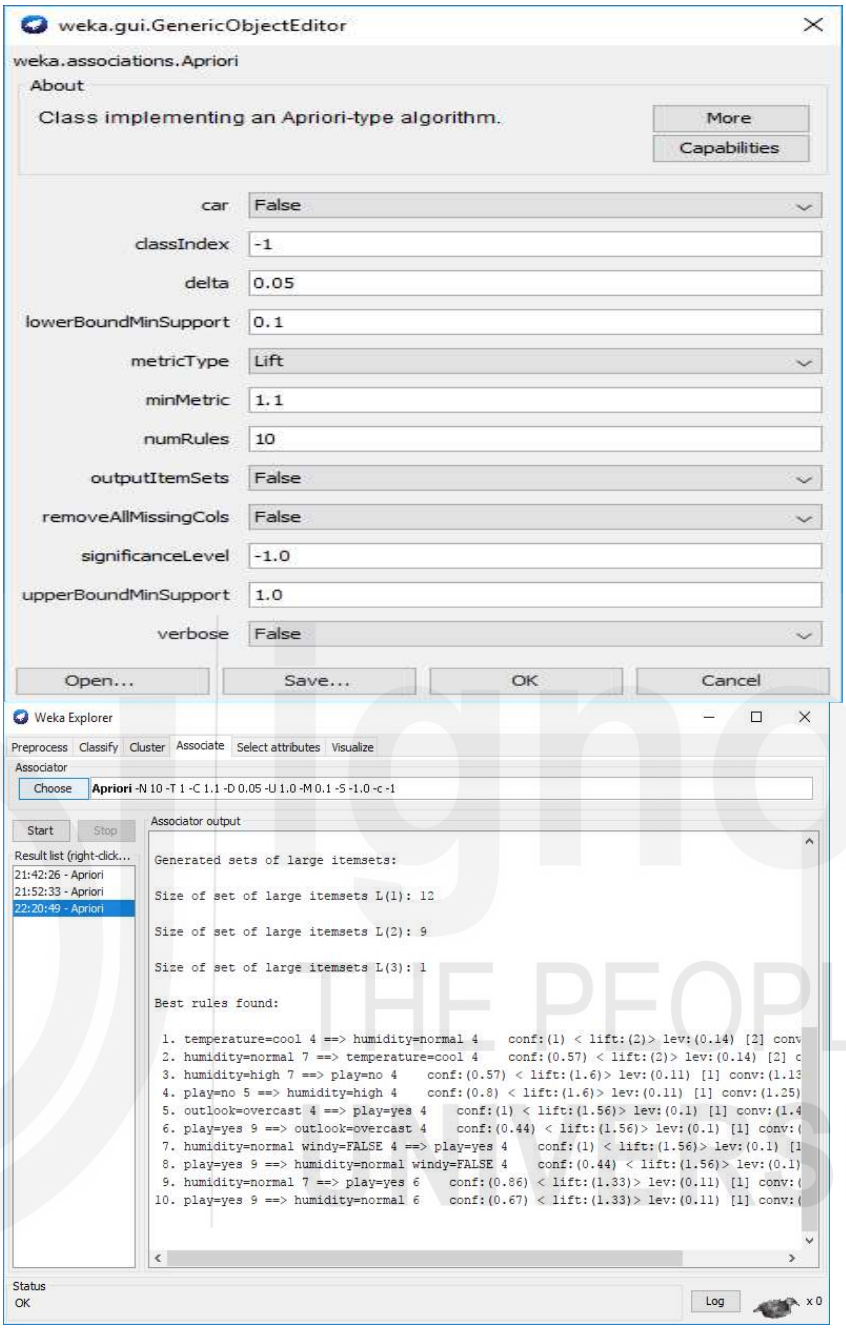
=== Associator model (full training set) ===
Minimum support: 0.15 (2 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 17
Generated sets of large item sets:
Size of set of large itemsetsL(1): 12
Size of set of large itemsetsL(2): 47
Size of set of large itemsetsL(3): 39
Size of set of large itemsetsL(4): 6
Best rules found:
1. outlook=overcast 4 ==> play=yes 4 conf:(1)
2. temperature=cool 4 ==> humidity=normal 4 conf:(1)
3. humidity=normal windy=FALSE 4 ==> play=yes 4 conf:(1)
4. outlook=sunny play=no 3 ==> humidity=high 3 conf:(1)
5. outlook=sunny humidity=high 3 ==> play=no 3 conf:(1)

6. outlook=rainy play=yes 3 ==> windy=FALSE 3 conf:(1)
7. outlook=rainy windy=FALSE 3 ==> play=yes 3 conf:(1)
8. temperature=cool play=yes 3 ==> humidity=normal 3 conf:(1)
9. outlook=sunny temperature=hot 2 ==> humidity=high 2 conf:(1)
10. temperature=hot play=no 2 ==> outlook=sunny 2 conf:(1)

=== Run information ===
Scheme: weka.associations.Apriori -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation: weather.symbolic
Instances: 14
Attributes: 5
outlook
temperature
humidity
windy
play

=== Associator model (full training set) ===
Minimum support: 0.15 (2 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 17
Generated sets of large item sets:
Size of set of large itemsetsL(1): 12
Size of set of large itemsetsL(2): 47
Size of set of large itemsetsL(3): 39
Size of set of large itemsetsL(4): 6
Best rules found:
1. outlook=overcast 4 ==> play=yes 4 conf:(1)
2. temperature=cool 4 ==> humidity=normal 4 conf:(1)
3. humidity=normal windy=FALSE 4 ==> play=yes 4 conf:(1)
4. outlook=sunny play=no 3 ==> humidity=high 3 conf:(1)
5. outlook=sunny humidity=high 3 ==> play=no 3 conf:(1)
6. outlook=rainy play=yes 3 ==> windy=FALSE 3 conf:(1)
7. outlook=rainy windy=FALSE 3 ==> play=yes 3 conf:(1)
8. temperature=cool play=yes 3 ==> humidity=normal 3 conf:(1)
9. outlook=sunny temperature=hot 2 ==> humidity=high 2 conf:(1)
10. temperature=hot play=no 2 ==> outlook=sunny 2 conf:(1)

(b) Using 10 numrules by using Lift

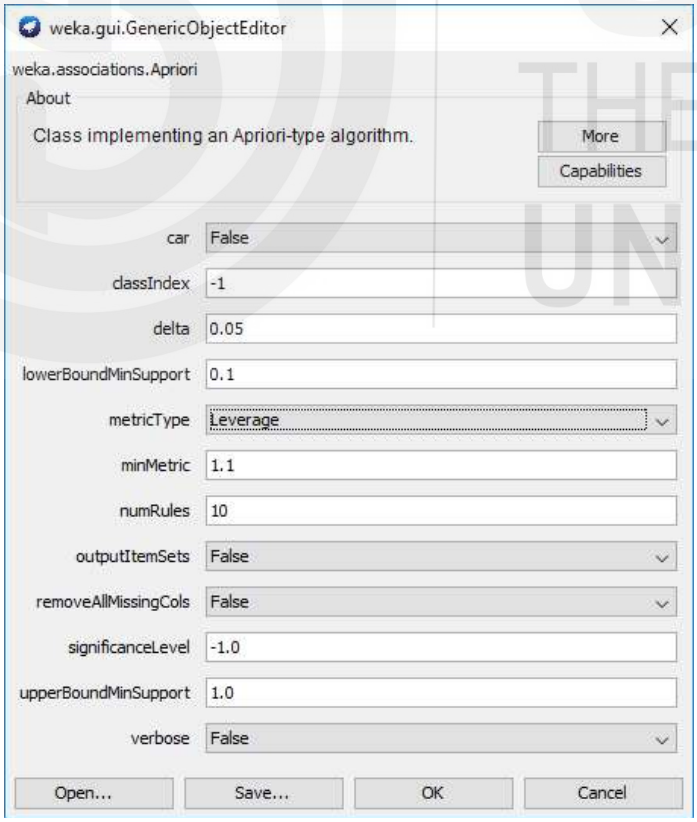


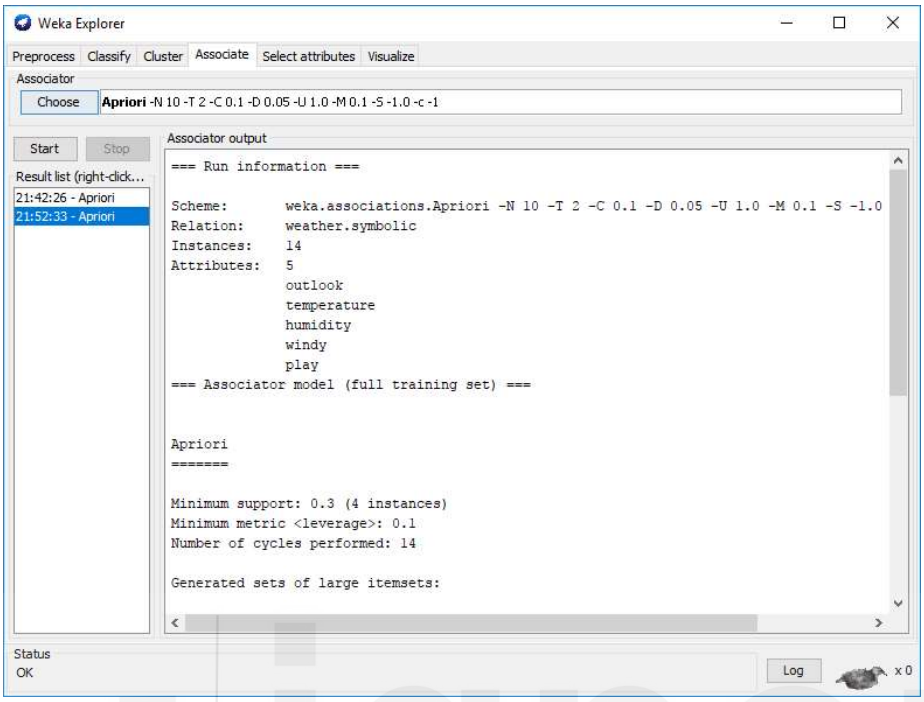
=== Run information ===
Scheme: weka.associations.Apriori -N 10 -T 1 -C 1.1 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation: weather.symbolic
Instances: 14
Attributes: 5

=== Associator model (full training set) ===
Minimum support: 0.3 (4 instances)
Minimum metric <lift>: 1.1
Number of cycles performed: 14
Generated sets of large item sets:
Size of set of large itemsetsL(1): 12

Size of set of large itemsetsL(2): 9
Size of set of large itemsetsL(3): 1
Best rules found:
1. temperature=cool 4 ==> humidity=normal 4 conf:(1) < lift:(2)>lev:(0.14) [2] conv:(2)
2. humidity=normal 7 ==> temperature=cool 4 conf:(0.57) < lift:(2)>lev:(0.14) [2] conv:(1.25)
3. humidity=high 7 ==> play=no 4 conf:(0.57) < lift:(1.6)>lev:(0.11) [1] conv:(1.13)
4. play=no 5 ==> humidity=high 4 conf:(0.8) < lift:(1.6)>lev:(0.11) [1] conv:(1.25)
5. outlook=overcast 4 ==> play=yes 4 conf:(1) < lift:(1.56)>lev:(0.1) [1] conv:(1.43)
6. play=yes 9 ==> outlook=overcast 4 conf:(0.44) < lift:(1.56)>lev:(0.1) [1] conv:(1.07)
7. humidity=normal windy=FALSE 4 ==> play=yes 4 conf:(1) < lift:(1.56)>lev:(0.1) [1] conv:(1.43)
8. play=yes 9 ==> humidity=normal windy=FALSE 4 conf:(0.44) < lift:(1.56)>lev:(0.1) [1] conv:(1.07)
9. humidity=normal 7 ==> play=yes 6 conf:(0.86) < lift:(1.33)>lev:(0.11) [1] conv:(1.25)
10. play=yes 9 ==> humidity=normal 6 conf:(0.67) < lift:(1.33)>lev:(0.11) [1] conv:(1.13)

(c) Using 10 numrules by using Leverage



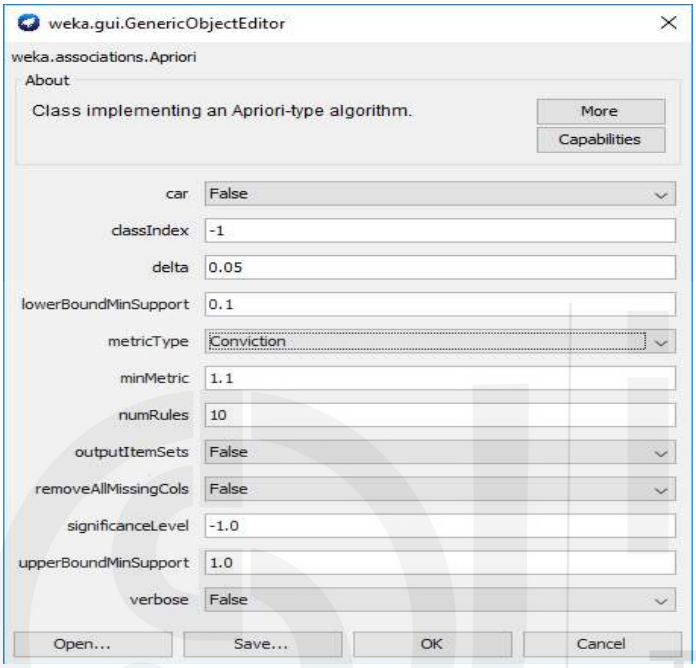


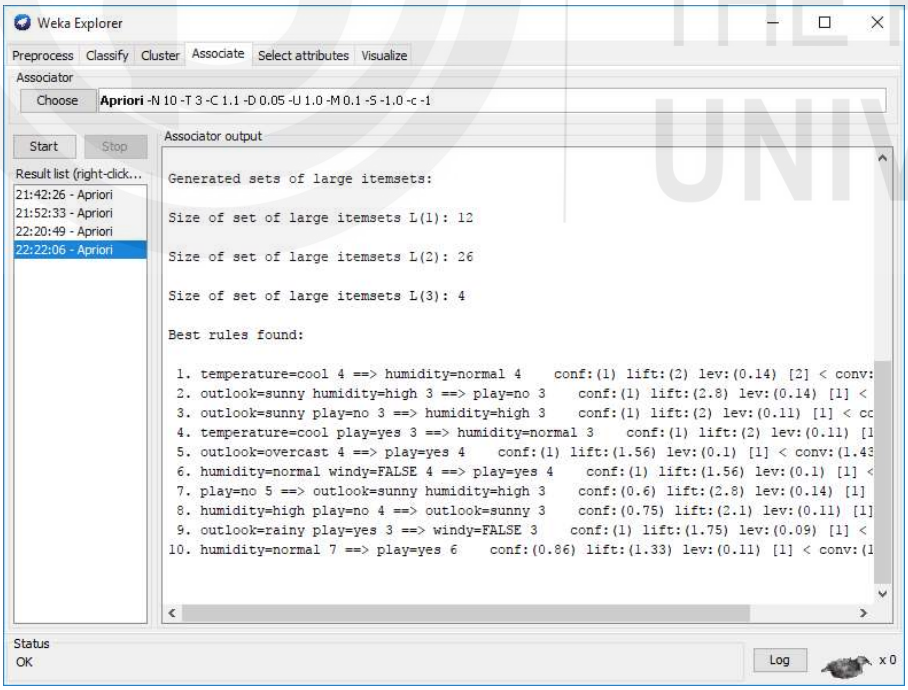
=== Run information ===
Scheme: weka.associations.Apriori -N 10 -T 2 -C 0.1 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation: weather.symbolic
Instances: 14
Attributes: 5

=== Associator model (full training set) ===
Minimum support: 0.3 (4 instances)
Minimum metric <leverage>: 0.1
Number of cycles performed: 14
Generated sets of large item sets:
Size of set of large itemsetsL(1): 12
Size of set of large itemsetsL(2): 9
Size of set of large itemsetsL(3): 1
Best rules found:
1. temperature=cool 4 ==> humidity=normal 4 conf:(1) lift:(2) <lev:(0.14) [2]>conv:(2)
2. humidity=normal 7 ==> temperature=cool 4 conf:(0.57) lift:(2) <lev:(0.14) [2]>conv:(1.25)
3. humidity=normal 7 ==> play=yes 6 conf:(0.86) lift:(1.33) <lev:(0.11) [1]>conv:(1.25)
4. play=yes 9 ==> humidity=normal 6 conf:(0.67) lift:(1.33) <lev:(0.11) [1]>conv:(1.13)
5. humidity=high 7 ==> play=no 4 conf:(0.57) lift:(1.6) <lev:(0.11) [1]>conv:(1.13)
6. play=no 5 ==> humidity=high 4 conf:(0.8) lift:(1.6) <lev:(0.11) [1]>conv:(1.25)
7. outlook=overcast 4 ==> play=yes 4 conf:(1) lift:(1.56) <lev:(0.1) [1]>conv:(1.43)

8. play=yes 9 ==> outlook=overcast 4 conf:(0.44) lift:(1.56) <lev:(0.1) [1]>conv:(1.07)
9. humidity=normal windy=FALSE 4 ==> play=yes 4 conf:(1) lift:(1.56) <lev:(0.1) [1]>conv:(1.43)
10. play=yes 9 ==> humidity=normal windy=FALSE 4 conf:(0.44) lift:(1.56) <lev:(0.1) [1]>conv:(1.07)

(d) Using 10 numrules by using Conviction





=== Run information ===
Scheme: weka.associations.Apriori -N 10 -T 3 -C 1.1 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1
Relation: weather.symbolic

Instances: 14
Attributes: 5

=== Associator model (full training set) ===
Minimum support: 0.25 (3 instances)
Minimum metric <conviction>: 1.1
Number of cycles performed: 15
Generated sets of large item sets:
Size of set of large itemsetsL(1): 12
Size of set of large itemsetsL(2): 26
Size of set of large itemsetsL(3): 4
Best rules found:
1. temperature=cool 4 ==> humidity=normal 4 conf:(1) lift:(2) lev:(0.14) [2] <conv:(2)>
2. outlook=sunny humidity=high 3 ==> play=no 3 conf:(1) lift:(2.8) lev:(0.14) [1] <conv:(1.93)>
3. outlook=sunny play=no 3 ==> humidity=high 3 conf:(1) lift:(2) lev:(0.11) [1] <conv:(1.5)>
4. temperature=cool play=yes 3 ==> humidity=normal 3 conf:(1) lift:(2) lev:(0.11) [1] <conv:(1.5)>
5. outlook=overcast 4 ==> play=yes 4 conf:(1) lift:(1.56) lev:(0.1) [1] <conv:(1.43)>
6. humidity=normal windy=FALSE 4 ==> play=yes 4 conf:(1) lift:(1.56) lev:(0.1) [1] <conv:(1.43)>
7. play=no 5 ==> outlook=sunny humidity=high 3 conf:(0.6) lift:(2.8) lev:(0.14) [1] <conv:(1.31)>
8. humidity=high play=no 4 ==> outlook=sunny 3 conf:(0.75) lift:(2.1) lev:(0.11) [1] <conv:(1.29)>
9. outlook=rainy play=yes 3 ==> windy=FALSE 3 conf:(1) lift:(1.75) lev:(0.09) [1] <conv:(1.29)>
10. humidity=normal 7 ==> play=yes 6 conf:(0.86) lift:(1.33) lev:(0.11) [1] <conv:(1.25)>

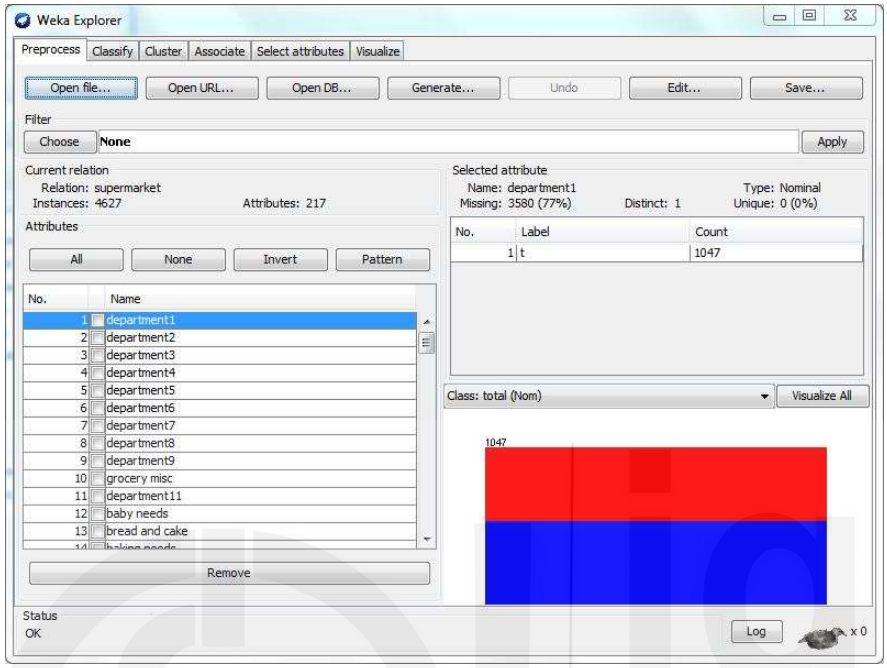
2.9.2 FP-TREE

FP-Tree: The FP-Tree algorithm is used to identify frequent patterns or frequent item sets in the area of Data Mining.

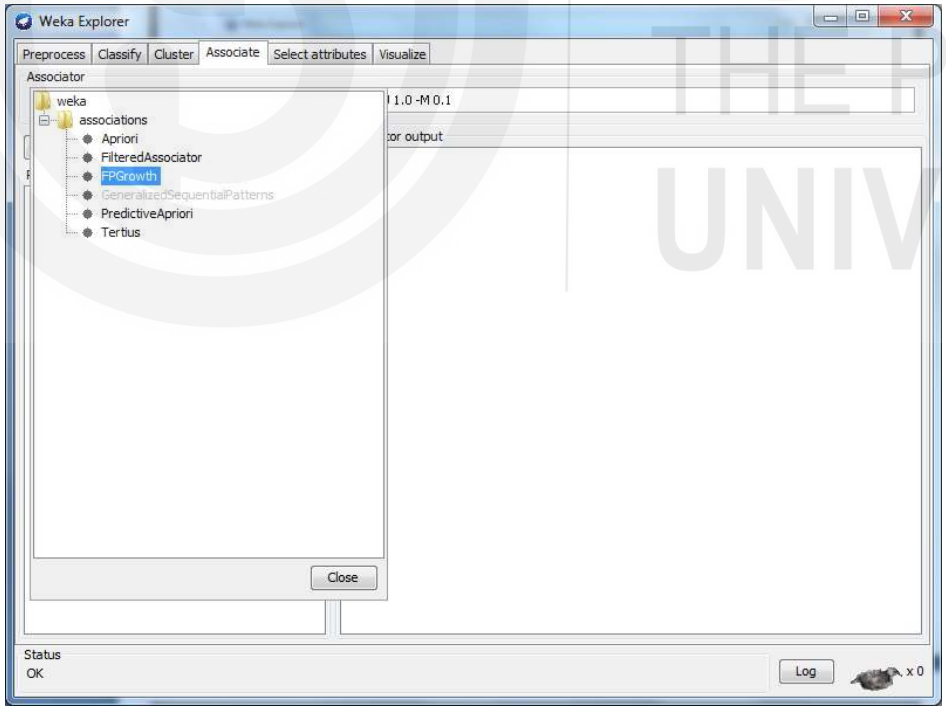
FP-Growth: The FP-Growth algorithm is an efficient and scalable method for mining the complete set of frequent patterns by pattern fragment growth, using an extended prefix-structure for storing compressed and crucial information about FP-Tree.

This was proposed by Han, J.Pei, and Y. Yin: Mining frequent patterns without candidate generation. In: Proceedings of the 2000 ACM-SIGMID International Conference on Management of Data, 1-12, 2000.

How to open the FP-TREE Algorithm in weka?
Start→weka→ select the explorer→choose(Super Market).



Selecting FP Tree algorithm in Associate menu



There are 4 metric types:

1.Confidence **2.**Lift **3.**Leverage **4.**Conviction

Useful Concepts :

To select interesting rules from the set of all possible rules, constraints on various measures of significance and interest can be used. The best-known constraints are minimum thresholds on support and confidence.

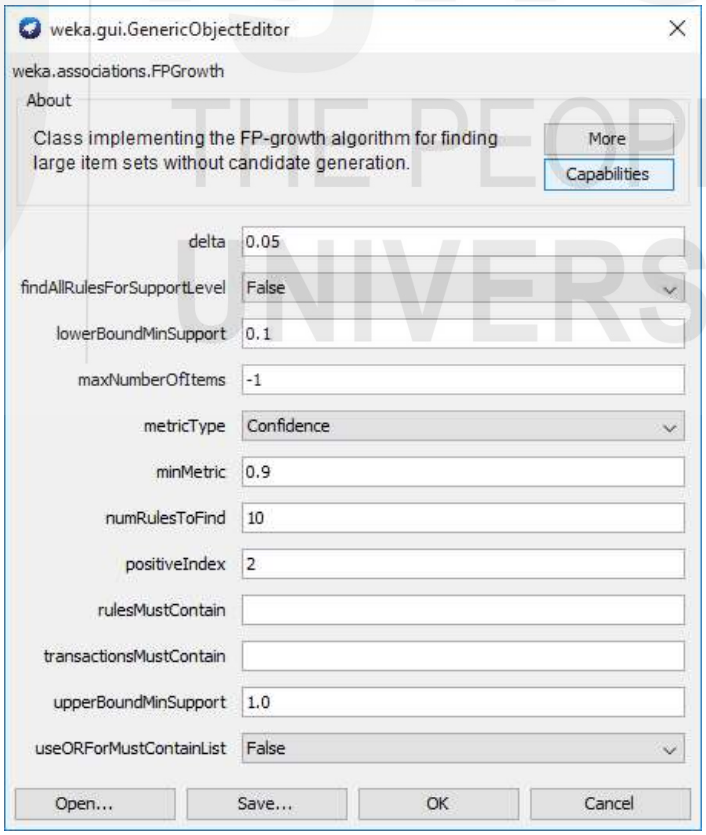
1. Support :
The support supp(X) of an item set X is defined as the proportion of transactions in the data set which contain the item set.
supp(X)=no of transactions which contain the item set X/total no of transactions

2. Confidence :
The confidence of a rule is defined
Conf(x->y) = supp(XUY)/supp(x)

3. Lift :
The lift of a rule is defined as:
lift(X->Y)= supp(XUY)/supp(Y)*supp(X)

4. Conviction :
The conviction of a rule is defined as:
conv(X_>Y)=1-supp(Y)/1-conf(X->Y)

Options:



Delta : Iteratively decrease support by this factor. Reduces support until min support is reached or required number of rules has been generated.

FindAllRulesForSupportLevel: Find all rules that meet the lower bound on minimum support and the minimum metric constraint. Turning this mode on will disable the iterative support reduction procedure to find the specified number of rules.

LowerBoundMinSupport : Lower bound for minimum support as a fraction or number of instances.

MaxNumberOfItems -- The maximum number of items to include in frequent item sets. -1 means no limit.

MetricType: Set the type of metric by which to rank rules. Confidence is the proportion of the examples covered by the premise that are also covered by the consequence(Class association rules can only be mined using confidence). Lift is confidence divided by the proportion of all examples that are covered by the consequence. This is a measure of the importance of the association that is independent of support. Leverage is the proportion of additional examples covered by both the premise and consequence above those expected if the premise and consequence were independent of each other. The total number of examples that this represents is presented in brackets following the leverage. Conviction is another measure of departure from independence.

MinMetric : Minimum metric score. Consider only rules with scores higher than this value.

NumRulesToFind : The number of rules to output

positiveIndex: Set the index of binary valued attributes that is to be considered the positive index. Has no effect for sparse data (in this case the first index (i.e. non-zero values) is always treated as positive. Also has no effect for unary valued attributes (i.e. when using the WekaApriori-style format for market basket data, which uses missing value "?" to indicate absence of an item.

RulesMustContain : Only print rules that contain these items. Provide a comma separated list of attribute names.

TransactionsMustContain : Limit input to FPGrowth to those transactions (instances) that contain these items. Provide a comma separated list of attribute names.

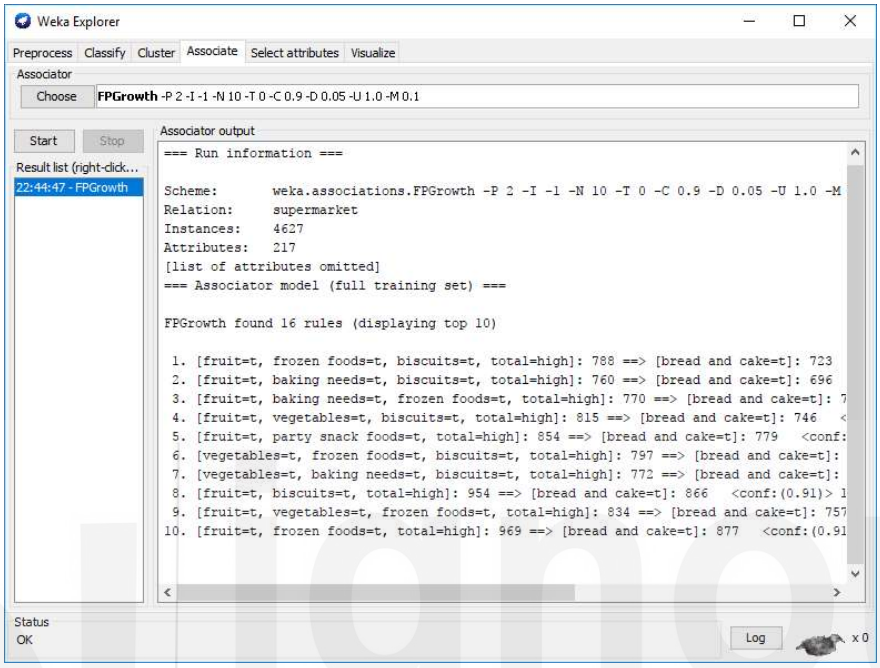
UpperBoundMinSupport : Upper bound for minimum support as a fraction or number of instances. Start iteratively decreasing minimum support from this value.

UseORForMustContainList : Use OR instead of AND for transactions/rules must contain lists.

NumRulesToFind: The number of rules to output

Select the Association→chooses the algorithm(FPTREE)→click on start.

Using 10 numrules by using confidence:



=== Run information ===
Scheme: weka.associations.FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1
Relation: supermarket
Instances: 4627
Attributes: 217
[list of attributes omitted]

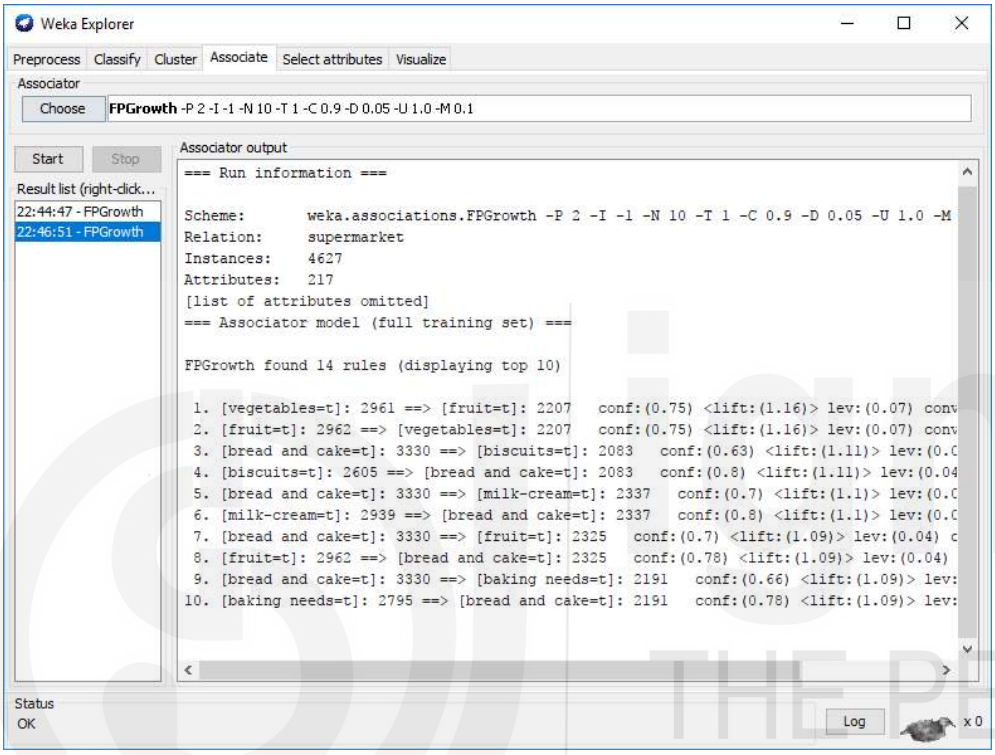
=== Associator model (full training set) ===

FPGrowth found 16 rules (displaying top 10)

1. [fruit=t, frozen foods=t, biscuits=t, total=high]: 788 ==> [bread and cake=t]: 723 <conf:(0.92)> lift:(1.27) lev:(0.03) conv:(3.35)
2. [fruit=t, baking needs=t, biscuits=t, total=high]: 760 ==> [bread and cake=t]: 696 <conf:(0.92)> lift:(1.27) lev:(0.03) conv:(3.28)
3. [fruit=t, baking needs=t, frozen foods=t, total=high]: 770 ==> [bread and cake=t]: 705 <conf:(0.92)> lift:(1.27) lev:(0.03) conv:(3.27)
4. [fruit=t, vegetables=t, biscuits=t, total=high]: 815 ==> [bread and cake=t]: 746 <conf:(0.92)> lift:(1.27) lev:(0.03) conv:(3.26)
5. [fruit=t, party snack foods=t, total=high]: 854 ==> [bread and cake=t]: 779 <conf:(0.91)> lift:(1.27) lev:(0.04) conv:(3.15)
6. [vegetables=t, frozen foods=t, biscuits=t, total=high]: 797 ==> [bread and cake=t]: 725 <conf:(0.91)> lift:(1.26) lev:(0.03) conv:(3.06)
7. [vegetables=t, baking needs=t, biscuits=t, total=high]: 772 ==> [bread and cake=t]: 701 <conf:(0.91)> lift:(1.26) lev:(0.03) conv:(3.01)

8. [fruit=t, biscuits=t, total=high]: 954 ==> [bread and cake=t]: 866 <conf:(0.91)> lift:(1.26) lev:(0.04) conv:(3)
9. [fruit=t, vegetables=t, frozen foods=t, total=high]: 834 ==> [bread and cake=t]: 757 <conf:(0.91)> lift:(1.26) lev:(0.03) conv:(3)
10. [fruit=t, frozen foods=t, total=high]: 969 ==> [bread and cake=t]: 877 <conf:(0.91)> lift:(1.26) lev:(0.04) conv:(2.92)

Using 10 numrules by using Lift



=== Run information ===
Scheme: weka.associations.FPGrowth -P 2 -I -1 -N 10 -T 1 -C 0.9 -D 0.05 -U 1.0 -M 0.1
Relation: supermarket
Instances: 4627
Attributes: 217
[list of attributes omitted]

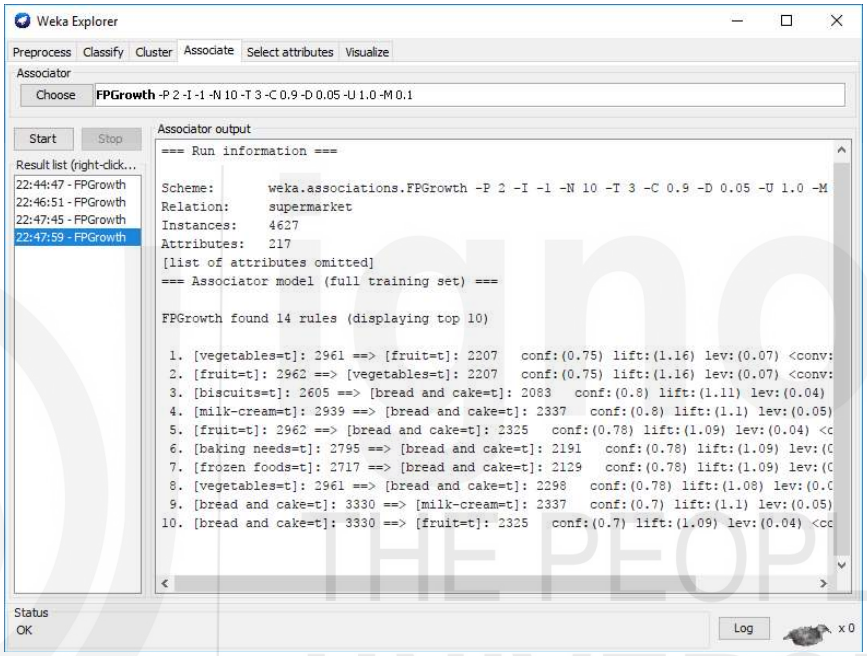
=== Associator model (full training set) ===

FPGrowth found 14 rules (displaying top 10)

1. [vegetables=t]: 2961 ==> [fruit=t]: 2207 conf:(0.75) <lift:(1.16)>lev:(0.07) conv:(1.41)
2. [fruit=t]: 2962 ==> [vegetables=t]: 2207 conf:(0.75) <lift:(1.16)>lev:(0.07) conv:(1.41)
3. [bread and cake=t]: 3330 ==> [biscuits=t]: 2083 conf:(0.63) <lift:(1.11)>lev:(0.04) conv:(1.17)
4. [biscuits=t]: 2605 ==> [bread and cake=t]: 2083 conf:(0.8) <lift:(1.11)>lev:(0.04) conv:(1.4)
5. [bread and cake=t]: 3330 ==> [milk-cream=t]: 2337 conf:(0.7) <lift:(1.1)>lev:(0.05) conv:(1.22)

6. [milk-cream=t]: 2939 ==> [bread and cake=t]: 2337 conf:(0.8) <lift:(1.1)>lev:(0.05) conv:(1.37)
7. [bread and cake=t]: 3330 ==> [fruit=t]: 2325 conf:(0.7) <lift:(1.09)>lev:(0.04) conv:(1.19)
8. [fruit=t]: 2962 ==> [bread and cake=t]: 2325 conf:(0.78) <lift:(1.09)>lev:(0.04) conv:(1.3)
9. [bread and cake=t]: 3330 ==> [baking needs=t]: 2191 conf:(0.66) <lift:(1.09)>lev:(0.04) conv:(1.16)
10. [baking needs=t]: 2795 ==> [bread and cake=t]: 2191 conf:(0.78) <lift:(1.09)>lev:(0.04) conv:(1.29)

Using 10 numrules by using conviction



=== Run information ===
Scheme: weka.associations.FPGrowth -P 2 -I -1 -N 10 -T 3 -C 0.9 -D 0.05 -U 1.0 -M 0.1
Relation: supermarket
Instances: 4627
Attributes: 217
[list of attributes omitted]
=== Associator model (full training set) ===

FPGrowth found 14 rules (displaying top 10)

1. [vegetables=t]: 2961 ==> [fruit=t]: 2207 conf:(0.75) lift:(1.16) lev:(0.07) <conv:(1.41)>
2. [fruit=t]: 2962 ==> [vegetables=t]: 2207 conf:(0.75) lift:(1.16) lev:(0.07) <conv:(1.41)>
3. [biscuits=t]: 2605 ==> [bread and cake=t]: 2083 conf:(0.8) lift:(1.11) lev:(0.04) <conv:(1.4)>
4. [milk-cream=t]: 2939 ==> [bread and cake=t]: 2337 conf:(0.8) lift:(1.1) lev:(0.05) <conv:(1.37)>

5. [fruit=t]: 2962 ==> [bread and cake=t]: 2325 conf:(0.78) lift:(1.09) lev:(0.04) <conv:(1.3)>
6. [baking needs=t]: 2795 ==> [bread and cake=t]: 2191 conf:(0.78) lift:(1.09) lev:(0.04) <conv:(1.29)>
7. [frozen foods=t]: 2717 ==> [bread and cake=t]: 2129 conf:(0.78) lift:(1.09) lev:(0.04) <conv:(1.29)>
8. [vegetables=t]: 2961 ==> [bread and cake=t]: 2298 conf:(0.78) lift:(1.08) lev:(0.04) <conv:(1.25)>
9. [bread and cake=t]: 3330 ==> [milk-cream=t]: 2337 conf:(0.7) lift:(1.1) lev:(0.05) <conv:(1.22)>
10. [bread and cake=t]: 3330 ==> [fruit=t]: 2325 conf:(0.7) lift:(1.09) lev:(0.04) <conv:(1.19)>

2.10 CLASSIFICATION

The concept of classification is basically distributing data among the various classes defined on a data set. Classification algorithms learn this form of distribution from a given set of training and then try to classify it correctly when it comes to test data for which the class is not specified. The values that specify these classes on the dataset are given a label name and are used to determine the class of data to be given during the test.

2.10.1 Classification Algorithms in WEKA

In this section let us discuss 5 classification algorithms in WEKA. Each algorithm that you are going to study covers briefly how it works, key algorithm parameters and its demonstration using the WEKA Explorer interface. The five classification algorithms are:

- Logistic Regression
- Naive Bayes
- Decision Tree
- k-Nearest Neighbors
- Support Vector Machines

A standard classification problem will be used to demonstrate each algorithm, specifically, the Ionosphere binary classification problem. This is a good dataset to demonstrate classification algorithms because the input variables are numeric and all have the same scale the problem only has two classes to discriminate.

Each instance describes the properties of radar returns from the atmosphere and the task is to predict whether or not there is structure in the ionosphere or not. There are 34 numerical input variables of generally the same scale. You can learn more about this dataset on the UCI Machine Learning Repository. Top results are in the order of 98% accuracy.

Start the Weka Explorer:

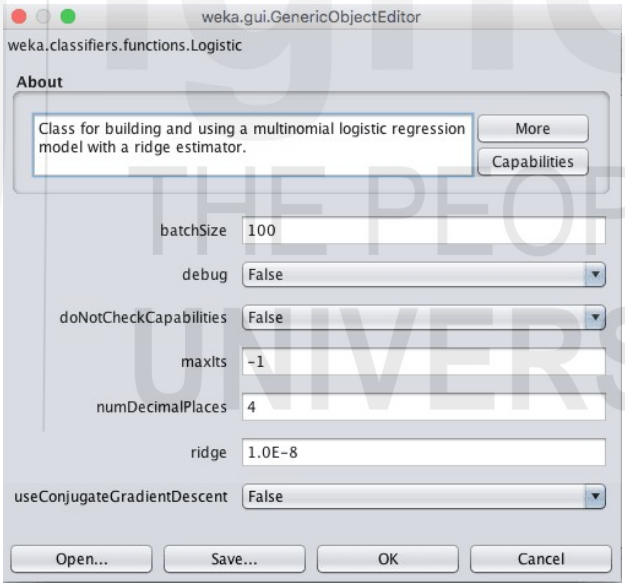
1. Open the Weka GUI Chooser.
2. Click the “Explorer” button to open the Weka Explorer.
3. Load the Ionosphere dataset from the *data/ionosphere.arff* file.
4. Click “Classify” to open the Classify tab.

2.10.1.1 Logistic Regression

Logistic regression is a binary classification algorithm.It assumes the input variables are numeric and have a Gaussian (bell curve) distribution. This last point does not have to be true, as logistic regression can still achieve good results if your data is not Gaussian. In the case of the Ionosphere dataset, some input attributes have a Gaussian-like distribution, but many do not.

The algorithm learns a coefficient for each input value, which are linearly combined into a regression function and transformed using a logistic (s-shaped) function. Logistic regression is a fast and simple technique, but can be very effective on some problems.
The logistic regression only supports binary classification problems, although the WEKA implementation has been adapted to support multi-class classification problems.
Choose the logistic regression algorithm:

1. Click the "Choose" button and select "Logistic" under the "functions" group.
2. Click on the name of the algorithm to review the algorithm configuration.

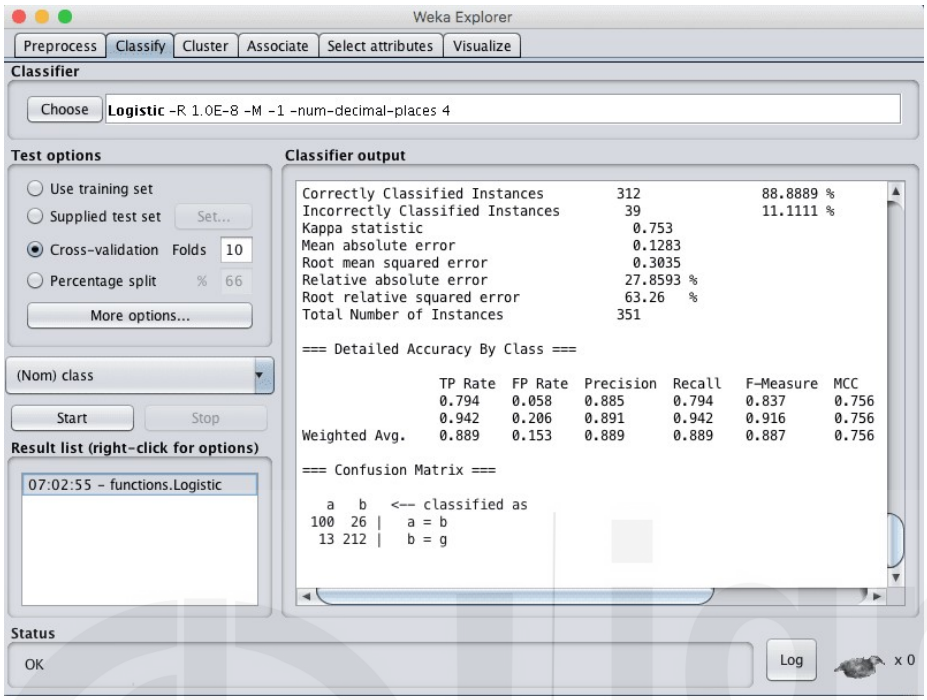


Weka Configuration for the Logistic Regression Algorithm

The algorithm can run for a fixed number of iterations (***maxIts***), but by default will run until it is estimated that the algorithm has converged. The implementation uses a ridge estimator which is a type of regularization. This method seeks to simplify the model during training by minimizing the coefficients learned by the model. The ridge parameter defines how much pressure to put on the algorithm to reduce the size of the coefficients. Setting this to 0 will turn off this regularization.

1. Click "OK" to close the algorithm configuration.
2. Click the "Start" button to run the algorithm on the Ionosphere dataset.

You can see that with the default configuration that logistic regression achieves an accuracy of 88%.



WEKA Classification Results for the Logistic Regression Algorithm

2.10.1.2 Naive Bayes

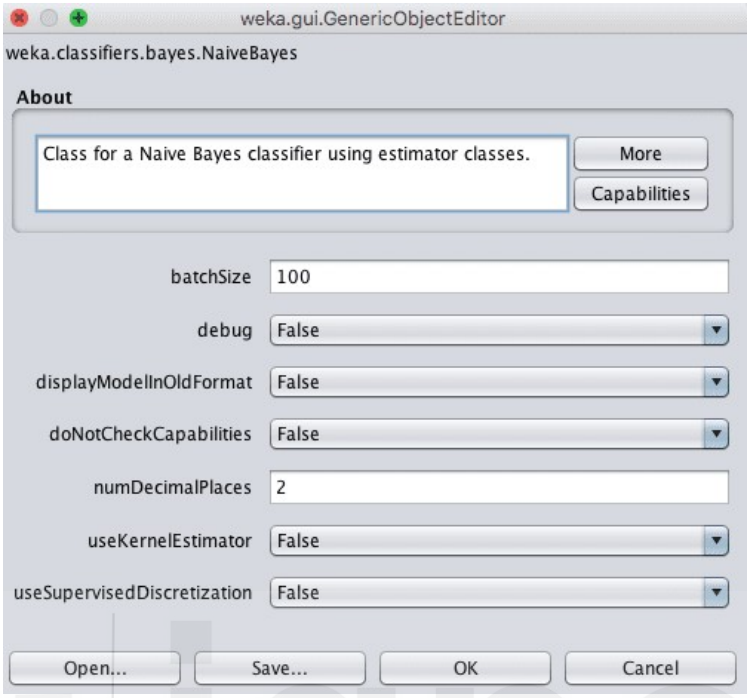
Naive Bayes is a classification algorithm. Traditionally it assumes that the input values are nominal, although it numerical inputs are supported by assuming a distribution.

Naive Bayes uses a simple implementation of Bayes Theorem (hence naive) where the prior probability for each class is calculated from the training data and assumed to be independent of each other (technically called conditionally independent).

This is an unrealistic assumption because we expect the variables to interact and be dependent, although this assumption makes the probabilities fast and easy to calculate. Even under this unrealistic assumption, Naive Bayes has been shown to be a very effective classification algorithm. Naive Bayes calculates the posterior probability for each class and makes a prediction for the class with the highest probability. As such, it supports both binary classification and multi-class classification problems.

Choose the Naive Bayes algorithm:

1. Click the “Choose” button and select “NaiveBayes” under the “bayes” group.
2. Click on the name of the algorithm to review the algorithm configuration.

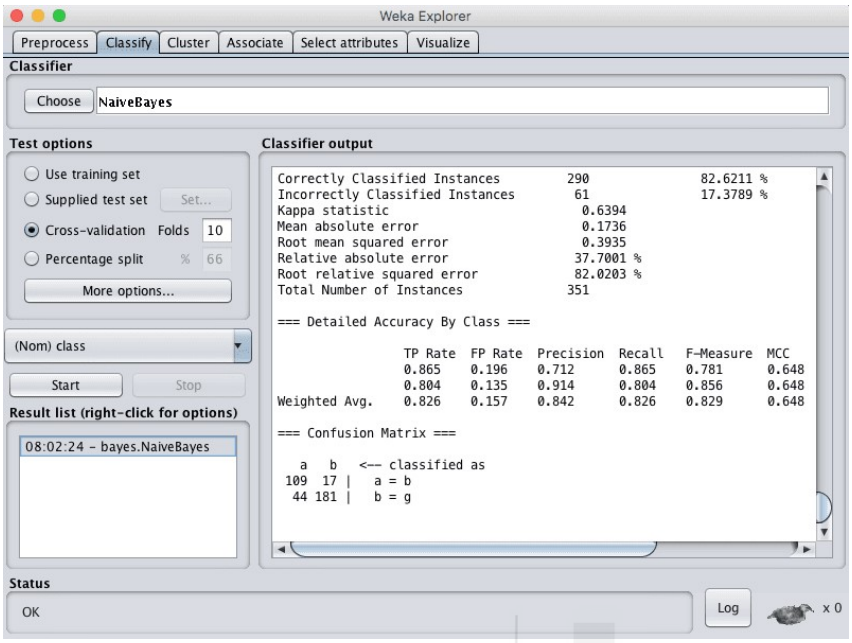


WEKA Configuration for the Naive Bayes Algorithm

By default a Gaussian distribution is assumed for each numerical attributes. You can change the algorithm to use a kernel estimator with the useKernelEstimator argument that may better match the actual distribution of the attributes in your dataset. Alternately, you can automatically convert numerical attributes to nominal attributes with the useSupervisedDiscretization parameter.

1. Click "OK" to close the algorithm configuration.
2. Click the "Start" button to run the algorithm on the Ionosphere dataset.
3.

You can see that with the default configuration that Naive Bayes achieves an accuracy of 82%.



WEKA Classification Results for the Naive Bayes Algorithm

2.10.1.3 Decision Tree

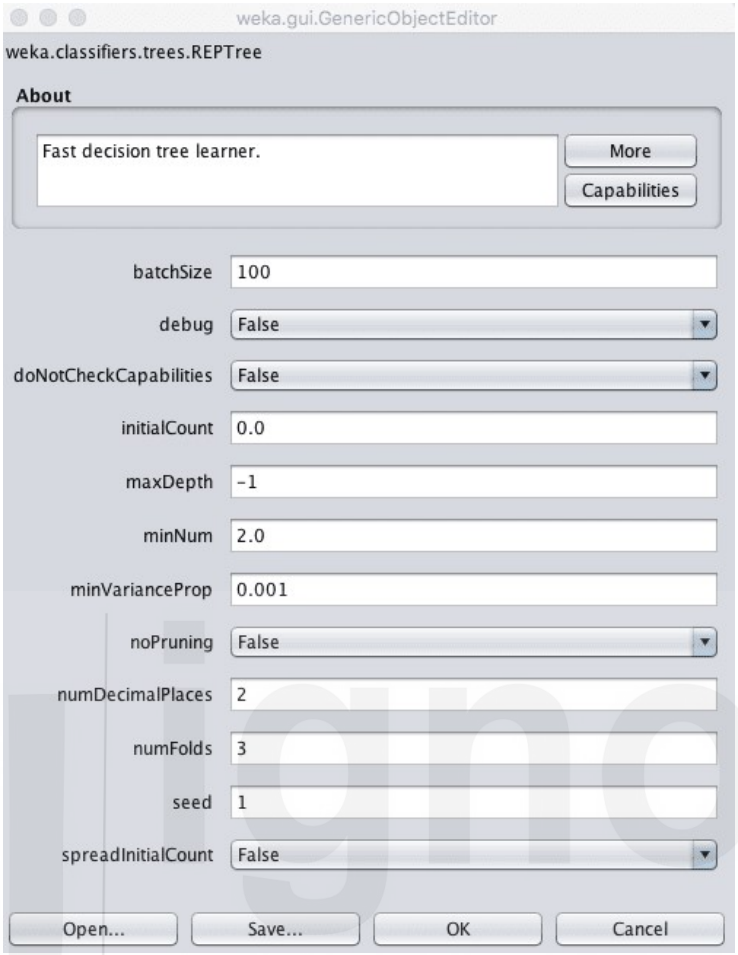
Decision trees can support classification and regression problems. Decision trees are more recently referred to as Classification And Regression Trees (CART).

They work by creating a tree to evaluate an instance of data, start at the root of the tree and moving town to the leaves (roots) until a prediction can be made. The process of creating a decision tree works by greedily selecting the best split point in order to make predictions and repeating the process until the tree is a fixed depth.

After the tree is constructed, it is pruned in order to improve the model's ability to generalize to new data.

Choose the decision tree algorithm:

1. Click the "Choose" button and select "REPTree" under the "trees" group.
2. Click on the name of the algorithm to review the algorithm configuration.



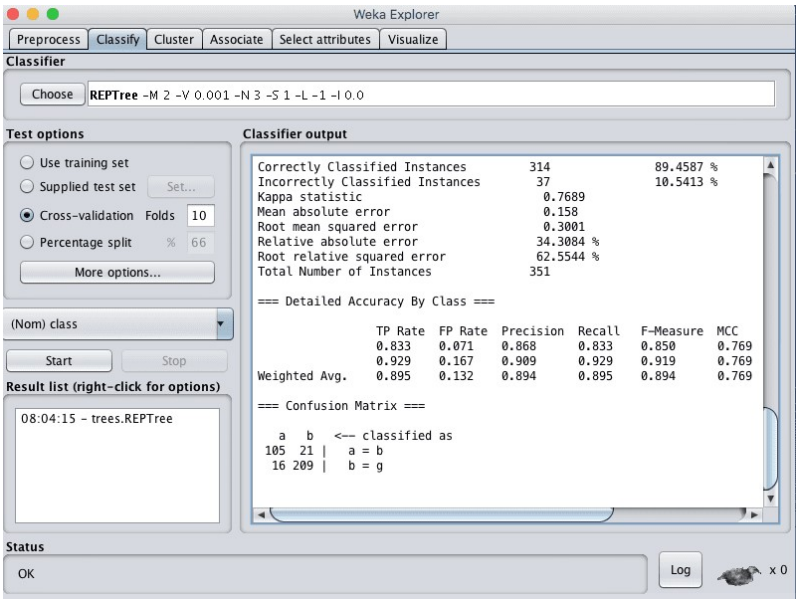
WEKA Configuration for the Decision Tree Algorithm

The depth of the tree is defined automatically, but a depth can be specified in the maxDepth attribute.

You can also choose to turn of pruning by setting the noPruning parameter to True, although this may result in worse performance. The minNum parameter defines the minimum number of instances supported by the tree in a leaf node when constructing the tree from the training data.

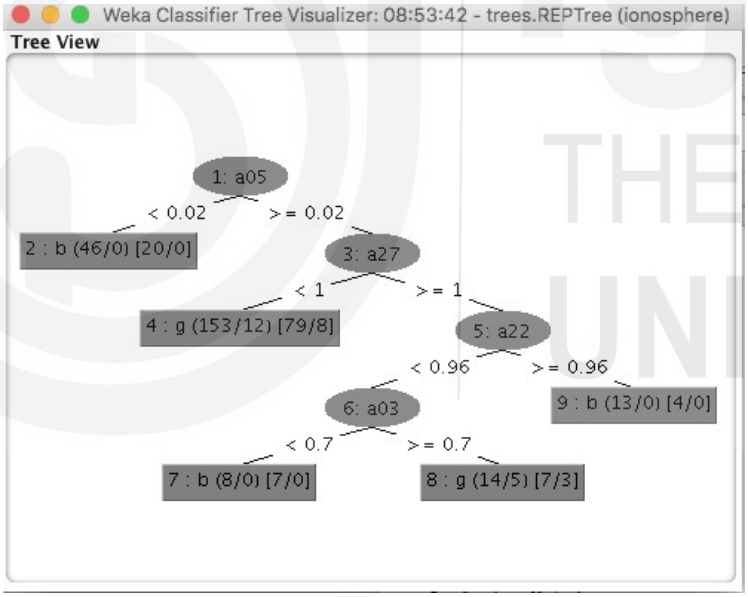
1. Click "OK" to close the algorithm configuration.
2. Click the "Start" button to run the algorithm on the Ionosphere dataset.

You can see that with the default configuration that the decision tree algorithm achieves an accuracy of 89%.



WEKA Classification Results for the Decision Tree Algorithm

Another more advanced decision tree algorithm that you can use is the C4.5 algorithm, called J48 in WEKA.You can review a visualization of a decision tree prepared on the entire training data set by right clicking on the "Result list" and clicking "Visualize Tree".



WEKA Visualization of a Decision Tree

2.10.1.4 k-Nearest Neighbors

The k-nearest neighbors algorithm supports both classification and regression. It is also called kNN for short. It works by storing the entire training dataset and querying it to locate the k most similar training patterns when making a prediction. As such, there is no model other than the raw training dataset and the only computation performed is the querying of the training dataset when a prediction is requested.

It is a simple algorithm, but one that does not assume very much about the problem other than that the distance between data instances is meaningful in making predictions. As such, it often achieves very good performance.

When making predictions on classification problems, KNN will take the mode (most common class) of the k most similar instances in the training dataset.

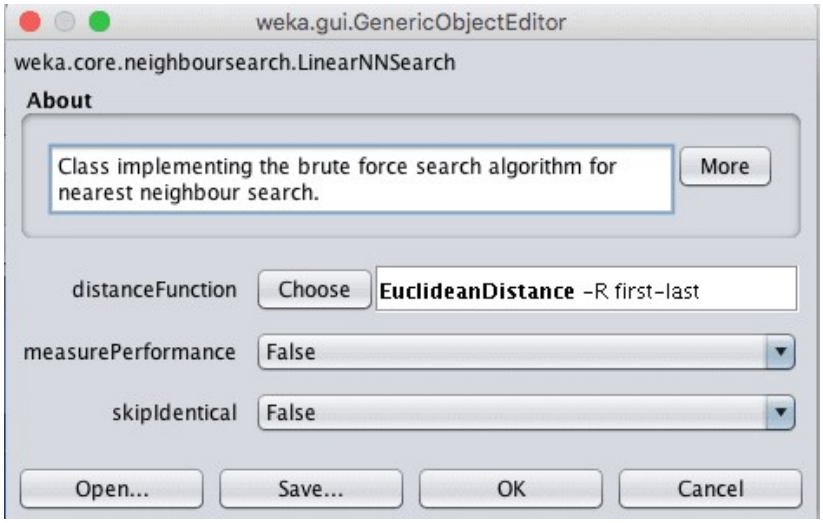
Choose the k-Nearest Neighbors algorithm:

1. Click the "Choose" button and select "IBk" under the "lazy" group.
2. Click on the name of the algorithm to review the algorithm configuration.

Weka Configuration for the k-Nearest Neighbors Algorithm

The size of the neighborhood is controlled by the k parameter. For example, if k is set to 1, then predictions are made using the single most similar training instance to a given new pattern for which a prediction is requested. Common values for k are 3, 7, 11 and 21, larger for larger dataset sizes. WEKA can automatically discover a good value for k using cross validation inside the algorithm by setting the crossValidate parameter to True. Another important parameter is the distance measure used. This is configured in the nearestNeighbourSearchAlgorithm which controls the way in which the training data is stored and searched.

The default is a LinearNNSearch. Clicking the name of this search algorithm will provide another configuration window where you can choose a distanceFunction parameter. By default, Euclidean distance is used to calculate the distance between instances, which is good for numerical data with the same scale. Manhattan distance is good to use if your attributes differ in measures or type.

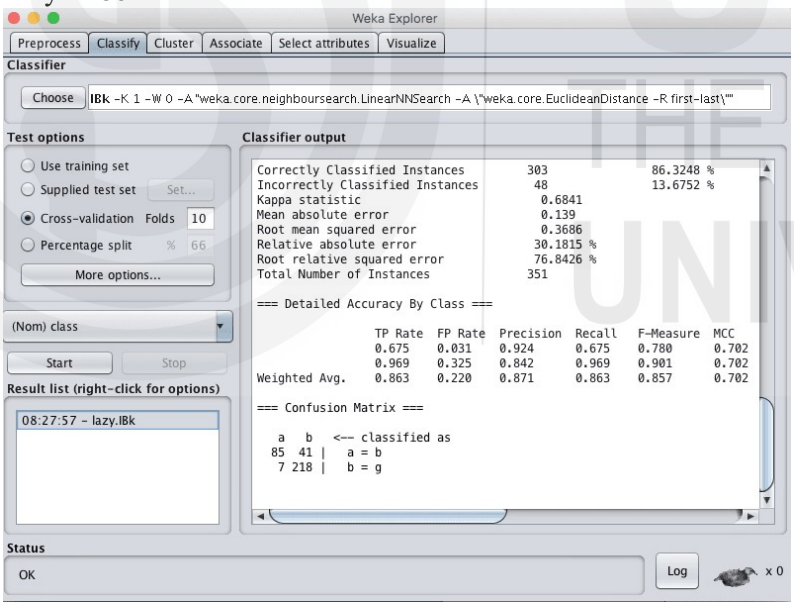


WEKA Configuration for the Search Algorithm in the k-Nearest Neighbors Algorithm

It is a good idea to try a suite of different k values and distance measures on your problem and see what works best.

1. Click "OK" to close the algorithm configuration.
2. Click the "Start" button to run the algorithm on the Ionosphere dataset.

You can see that with the default configuration that the kNN algorithm achieves an accuracy of 86%.



WEKA Classification Results for k-Nearest Neighbors

2.10.1.5 Support Vector Machines

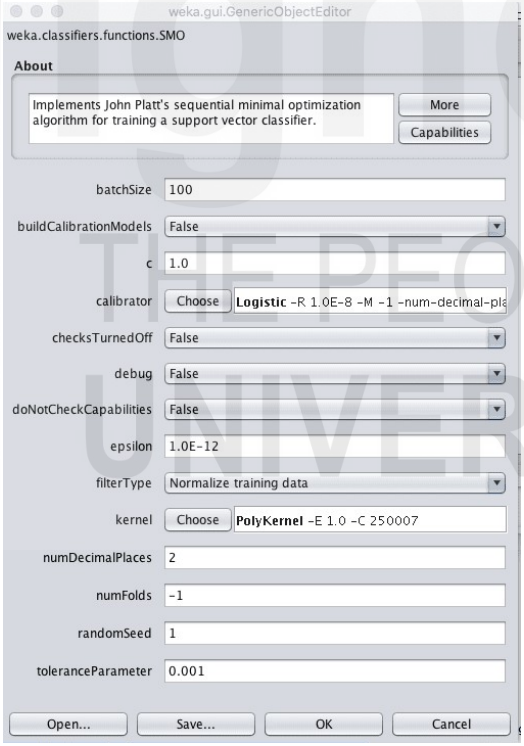
Support Vector Machines were developed for binary classification problems, although extensions to the technique have been made to support multi-class classification and regression problems. The algorithm is often referred to as SVM for short. SVM was developed for numerical input variables, although will automatically convert nominal values to numerical values. Input data is also normalized before being used.SVM work by finding a line that best separates the data into the two groups. This is done using an optimization

process that only considers those data instances in the training dataset that are closest to the line that best separates the classes. The instances are called support vectors, hence the name of the technique.In almost all problems of interest, a line cannot be drawn to neatly separate the classes, therefore a margin is added around the line to relax the constraint, allowing some instances to be misclassified but allowing a better result overall.

Finally, few datasets can be separated with just a straight line. Sometimes a line with curves or even polygonal regions need to be marked out. This is achieved with SVM by projecting the data into a higher dimensional space in order to draw the lines and make predictions. Different kernels can be used to control the projection and the amount of flexibility in separating the classes.
Choose the SVM algorithm:

1. Click the “Choose” button and select “SMO” under the “function” group.
2. Click on the name of the algorithm to review the algorithm configuration.

SMO refers to the specific efficient optimization algorithm used inside the SVM implementation, which stands for Sequential Minimal Optimization.



WEKA Configuration for the Support Vector Machines Algorithm

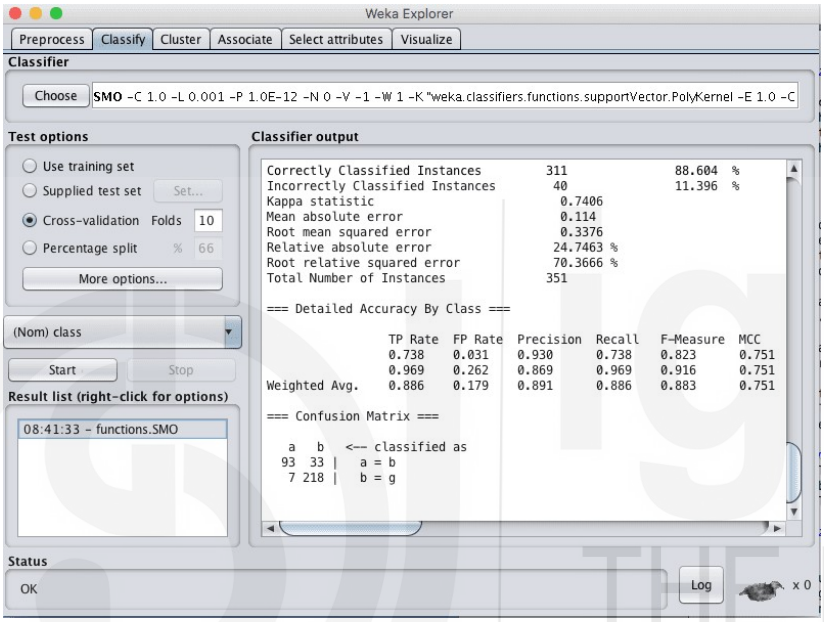
The C parameter, called the complexity parameter in Weka controls how flexible the process for drawing the line to separate the classes can be. A value of 0 allows no violations of the margin, whereas the default is 1.

A key parameter in SVM is the type of Kernel to use. The simplest kernel is a Linear kernel that separates data with a straight line or hyperplane. The default in Weka is a Polynomial Kernel that will separate the classes using a curved or wiggly line, the higher the polynomial, the more wiggly (the exponent value).

A popular and powerful kernel is the RBF Kernel or Radial Basis Function Kernel that is capable of learning closed polygons and complex shapes to separate the classes. It is a good idea to try a suite of different kernels and C (complexity) values on your problem and see what works best.

1. Click "OK" to close the algorithm configuration.
2. Click the "Start" button to run the algorithm on the Ionosphere dataset.

You can see that with the default configuration that the SVM algorithm achieves an accuracy of 88%.



WEKA Classification Results for the Support Vector Machine Algorithm

2.11 CLUSTERING

2.11.1 K-Means Algorithm Using WEKA Explorer

Let us see how to implement the K-means algorithm for clustering using WEKA Explorer.

Cluster Analysis

Clustering Algorithms are unsupervised learning algorithms used to create groups of data with similar characteristics. It aggregates objects with similarities into groups and subgroups thus leading to the partitioning of datasets. Cluster analysis is the process of portioning of datasets into subsets. These subsets are called clusters and the set of clusters is called clustering.
Cluster Analysis is used in many applications such as image recognition, pattern recognition, web search, and security, in business intelligence such as the grouping of customers with similar likings.

K-Means Clustering

K means clustering is the simplest clustering algorithm. In the K-Clustering algorithm, the dataset is partitioned into K-clusters. An objective function is used to find the quality of partitions so that similar objects are in one cluster and dissimilar objects in other groups.

In this method, the centroid of a cluster is found to represent a cluster. The centroid is taken as the center of the cluster which is calculated as the mean value of points within the cluster. Now the quality of clustering is found by measuring the Euclidean distance between the point and center. This distance should be maximum.

How Does K-Mean Clustering Algorithm Work?

Step 1: Choose a value of K where K is the number of clusters.

Step 2: Iterate each point and assign the cluster which is having the nearest center to it. When each element is iterated then compute the centroid of all the clusters.

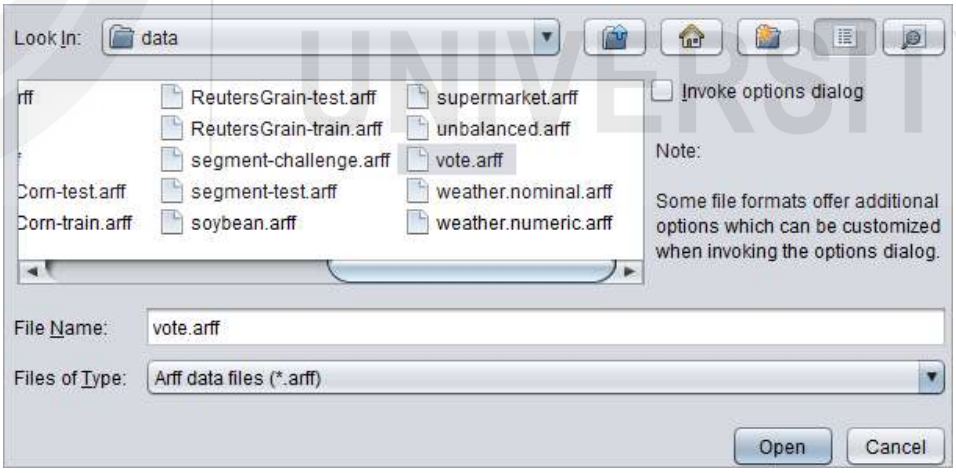
Step 3: Iterate every element from the dataset and calculate the Euclidean distance between the point and the centroid of every cluster. If any point is present in the cluster which is not nearest to it then reassign that point to the nearest cluster and after performing this to all the points in the dataset, again calculate the centroid of each cluster.

Step 4: Perform Step#3 until there is no new assignment that took place between the two consecutive iterations.

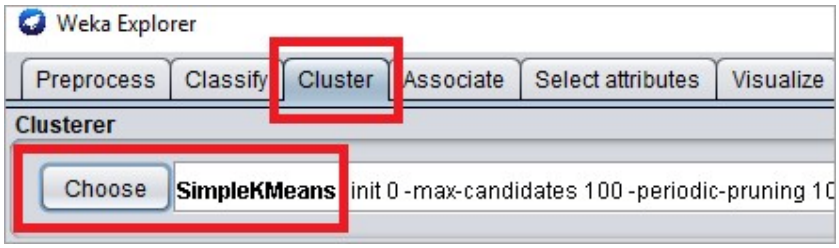
K-Means Clustering Implementation Using WEKA

The steps for implementation using WEKa are as follows:

1) Open WEKA Explorer and click on Open File in the Preprocess tab. Choose dataset "vote.arff".



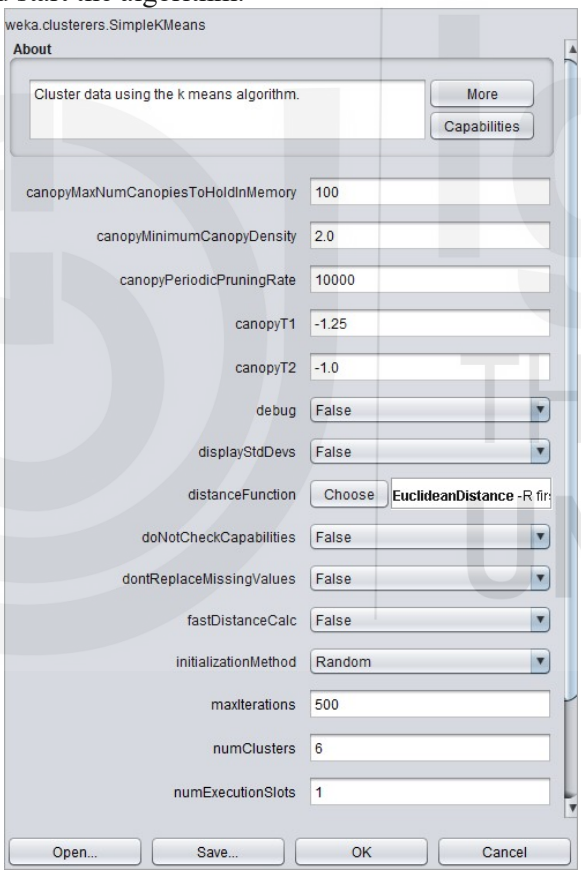
2) Go to the "Cluster" tab and click on the "Choose" button. Select the clustering method as "SimpleKMeans".



3) Choose Settings and then set the following fields:

- Distance function as Euclidian
- The number of clusters as 6. With more number of clusters, the sum of squared error will reduce.
- Seed as 10. of

Click on Ok and start the algorithm.

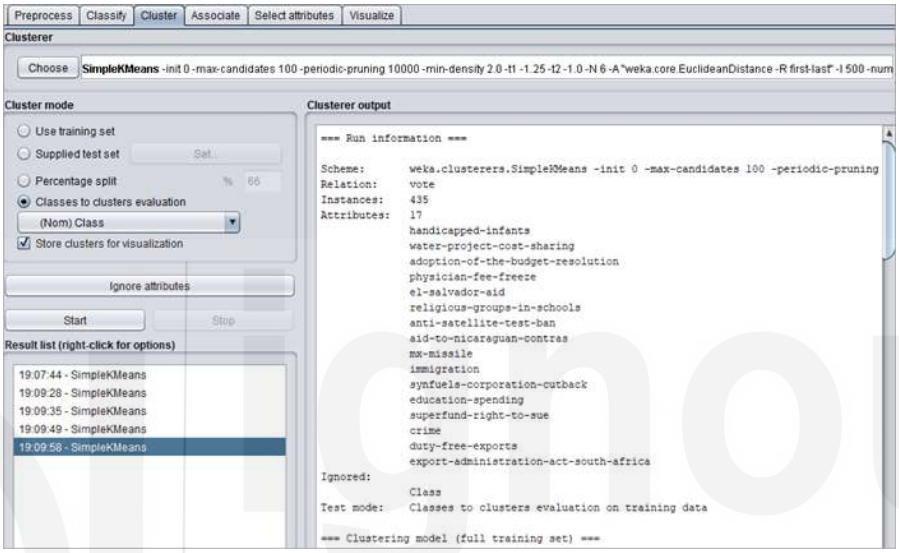


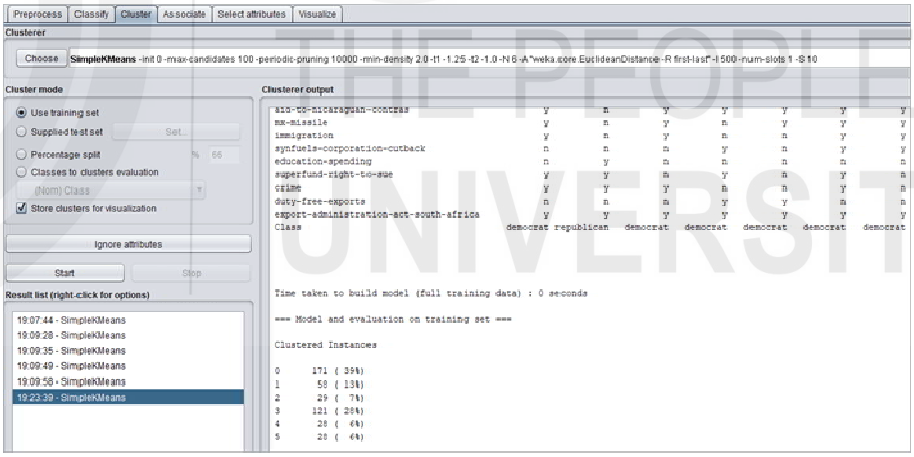
4) Click on Start in the left panel. The algorithm display results on the white screen.

Let us analyze the run information:

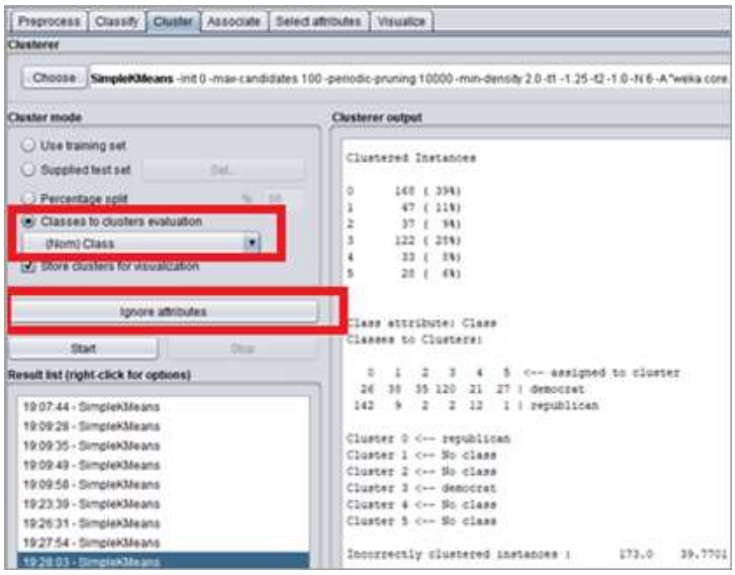
- Scheme, Relation, Instances, and Attributes describe the property of the dataset and the clustering method used. In this case, vote.arff dataset has 435 instances and 13 attributes.
- With the Kmeans cluster, the number of iterations is 5.

- The sum of the squared error is 1098.0. This error will reduce with an increase in the number of clusters.
- The 5 final clusters with centroids are represented in the form of a table. In our case, Centroids of clusters are 168.0, 47.0, 37.0, 122.0.33.0 and 28.0.
- Clustered instances represent the number and percentage of total instances falling in the cluster.





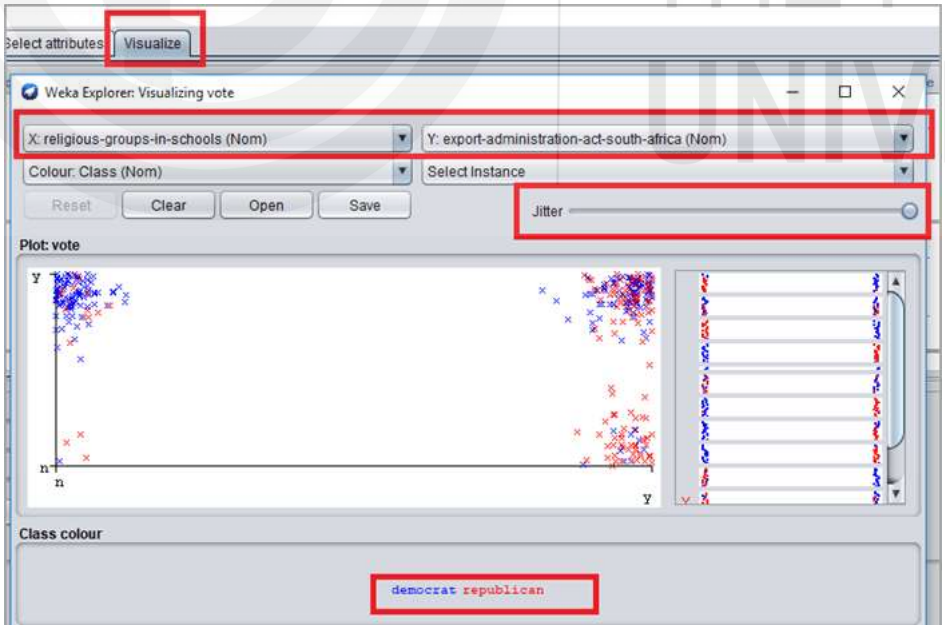
5) Choose "Classes to Clusters Evaluations" and click on Start.
The algorithm will assign the class label to the cluster. Cluster 0 represents republican and Cluster 3 represents democrat. The Incorrectly clustered instance is 39.77% which can be reduced by ignoring the unimportant attributes.



6) To ignore the unimportant attributes. Click on the "Ignore attributes" button and select the attributes to be removed.

#7) Use the "Visualize" tab to visualize the Clustering algorithm result. Go to the tab and click on any box. Move the Jitter to the max.

- The X-axis and Y-axis represent the attribute.
- The blue color represents class label democrat and the red color represents class label republican.
- Jitter is used to view Clusters.
- Click the box on the right-hand side of the window to change the x coordinate attribute and view clustering with respect to other attributes.



Output

K means clustering is a simple cluster analysis method. The number of clusters can be set using the setting tab. The centroid of each cluster is calculated as the mean of all points within the clusters. With the increase in the number of

clusters, the sum of square errors is reduced. The objects within the cluster exhibit similar characteristics and properties. The clusters represent the class labels.

2.11.2 Hierarchical Clustering

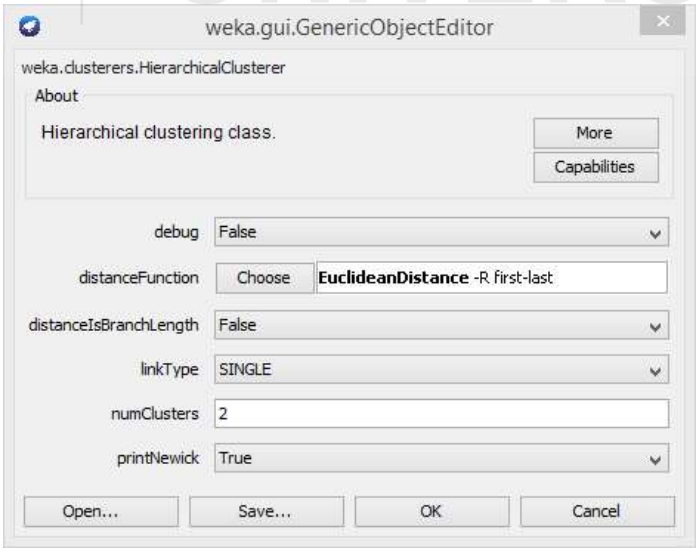
Hierarchical clustering class implements a number of classic Agglomorative (i.e. bottom up) hierarchical clustering methods.

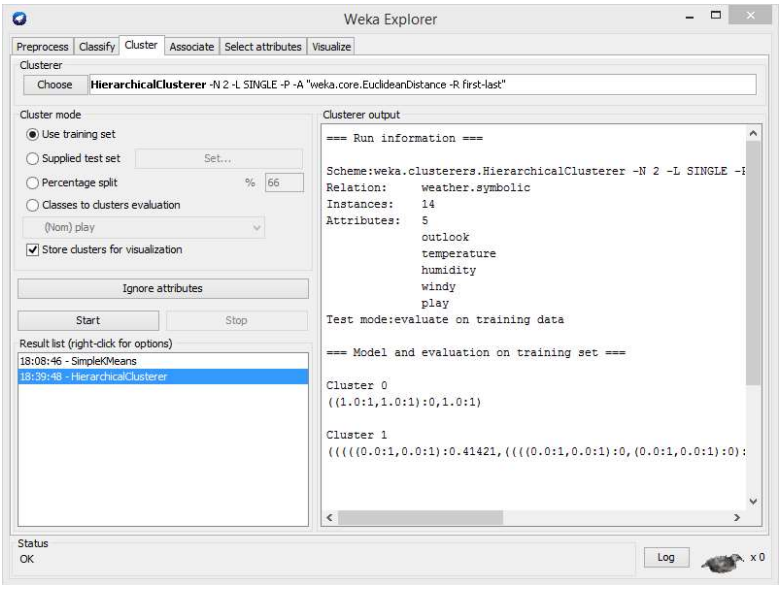
Options

debug -- If set to true, classifier may output additional info to the console.
distanceFunction -- Sets the distance function, which measures the distance between two individual. instances (or possibly the distance between an instance and the centroid of a clusterdepending on the Link type).
distanceIsBranchLength -- If set to false, the distance between clusters is interpreted as the height of the node linking the clusters. This is appropriate for example for single link clustering. However, for neighbor joining, the distance is better interpreted as branch length. Set this flag to get the latter interpretation.
linkType -- Sets the method used to measure the distance between two clusters.
numClusters -- Sets the number of clusters. If a single hierarchy is desired, set this to 1.
printNewick -- Flag to indicate whether the cluster should be print in Newick format. This can be useful for display in other programs. However, for large datasets a lot of text may be produced, which may not be a nuisance when the Newick format is not required.

Procedure

- Select the choose button in the cluster frame to choose a clustering technique.
- Choose appropriate mode in the cluster mode frame.
- Click on start button, the technique will be performed on the selected data.
- The output window appears as given below.





=== Run information ===
Scheme:weka.clusterers.HierarchicalClusterer -N 2 -L SINGLE -P -A "weka.core.EuclideanDistance -R first-last"
Relation: weather.symbolic
Instances: 14
Attributes: 5
outlook
temperature
humidity
windy
play
Test mode:evaluate on training data

=== Model and evaluation on training set ===

Cluster 0
((1.0:1,1.0:1):0,1.0:1)

Cluster 1
(((((0.0:1,0.0:1):0.41421,((((0.0:1,0.0:1):0,(0.0:1,0.0:1):0):0.41421,1.0:1.41421):0,0.0:1.41421):0):0,0.0:1.41421):0,0.0:1.41421):0,1.0:1.41421)

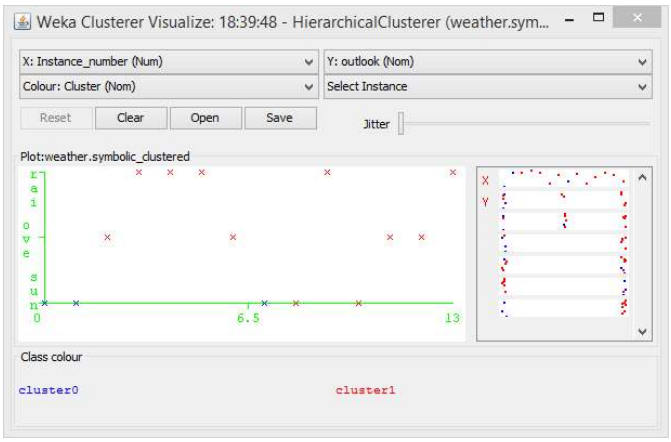
Time taken to build model (full training data) : 0.02 seconds

=== Model and evaluation on training set ===

Clustered Instances

0 3 (21%)
1 11 (79%)

Visualizing the Cluster Assignments



2.12 GENERAL GUIDELINES

Following are some of the general guidelines:

- Observation book and Lab record are compulsory.
- You should attempt all problems/assignments given in the list session wise.
- For the tasks related to the working with the WEKA, describe the procedure and also present screenshots wherever applicable.
- You may seek assistance in doing the lab exercises from the concerned lab instructor. Since the assignments have credits, the lab instructor is obviously not expected to tell you how to solve these, but you may ask questions concerning the Operating system and C programs.
- Add comments wherever necessary.
- The program should be interactive, general and properly documented with real Input/ Output data.
- If two or more submissions from different students appear to be of the same origin (i.e. are variants of essentially the same program), none of them will be counted. You are strongly advised not to copy somebody else's work.
- It is your responsibility to create a separate directory to store all the programs, so that nobody else can read or copy.
- The list of the programs(list of programs given at the end, session-wise) is available to you in this lab manual. For each session, you must come prepare with the algorithms and the programs written in the Observation Book. You should utilize the lab hours for executing the programs, testing for various desired outputs and enhancements of the programs.
- As soon as you have finished a lab exercise, contact one of the lab instructor / incharge in order to get the exercise evaluated and also get the signature from him/her on the Observation book.

- Completed lab assignments should be submitted in the form of a Lab Record in which you have to write the algorithm, program code along with comments and output for various inputs given.

- The total no. of lab sessions (3 hours each) are 10 and the list of assignments is provided session-wise. It is important to observe the deadline given for each assignment.

Get started with WEKAand explore the utilities sessionwise.

2.13 PRACTICAL SESSIONS

Following are the sessionwise practical problems. Attempt all these practical problems.

Session-1

1. Download and install WEKA. Navigate the various options available in WEKA. Explore the available datasets in WEKA. Load various datasets and observe the following"
    a. List the attribute names and their types
    b. No. of records in each dataset
    c. Identify the class attribute(if any)
    d. Plot Histogram
    e. Determine the no. of records for each class.
    f. Visualize the data in different dimensions.
2. Create your own EXCEL file. Convert the EXCEL file to ***.csv*** format and prepare it as ***.arff*** file.
3. Try to create your own datasets.
4. Preprocess and classify Customer, Argiculture, Weather, Whole-sale Customers or the datasets of your own choice from https://archive.ics.uci.edu/ml/datasets.php

Session-2

5. Perform the basic pre-processing operations on data relation such as removing an attribute and filter attribute bank data
6. Demonstrate the preprocessing with various options on the following datasets:
    a. student.arff
    b. labor.arff
    c. contactlenses.arff

Session-3

7. Perform the following:
   - Explore various options available in WEKA for preprocessing data and apply unsupervised filters like Discretization, Resample-filter etc.. on various datasets.
   - Load weather, nominal, Iris, Glass datasets into WEKA and run Apriori algorithms with different support and confidence values.
   - Study the rules generated.
   - Apply different discretization filters on numerical attributes and run the Apriori association rule algorithm. Study the rules generated.
   - Derive interesting insights and observe the effect of discretization in the rule generation process.
8. Implement the Apriori Algorithm to find the association rules in contactless.arff dataset.

Session-4

9. Find the frequent patterns using FP-Growth algorithm on contactlenses.arff and test.arff datasets.
10. Generate association rules using Apriori algorithm with Bank.arff relation
    a. Set minimum support range as 20% -100% incremental decrease factor as 5% and confidence factor as 80% and generate 5 rules.
    b. Set minimum support as 10%, delta 5%, minimum average(4ft) as 150% and generate 4 rules.
11. Generate association rule for the credit card promotion dataset using a priory algorithm with the support range 40% to 100% confidence as 10% incremental decrease as 5% and generate 6 rules.

Session-5

12. Perform the following:
    - Use contactlenses.arff and load it into WEKA. Check that all attributes are nominal (categorical).
    - Change to the Associate Panel. Select “Apriori” as associator. After pressing the start button, Apriori starts to build its model and writes its output into the output field. The first part of the output (“Run

information") describes the options that have been set and the data set used. Make sure you understand all the data reported.

- The rules that have been generated are listed at the end of the output. By default, only the 10 most valuable rules according to their confidence level are shown. Each rule consists of some attribute values on a left hand side of the arrow, the arrow sign and the right hand side list of attribute values. Right of the arrow sign are the predicted attribute values. Rules have certain support and confidence values. The number before the arrow sign is the number of instances the rule applies to. The number after the arrow sign is the number of instances predicted correctly. The number in brackets after 'conf:' is the confidence of the rule. Analyse the rules mined from the data set. What are their confidence and support values? Examine the number of large itemsets – make sure you understand how this data has been calculated (check that the values you would get 'manually' are correct).

Session-6

13. Perform the following:
    - Use zoo.arff dataset and load it into WEKA. Examine the attributes and make sure you understand their meaning. Are all attributes nominal?
    - In the preprocess area, deselect the animal and legs attributes. The animal attribute is the name of the animal, and is not useful for mining. The legs attribute is numeric and cannot be used directly with Apriori. Alternatively, you can try to use the Discretize Filter to discretize the legs attribute.
    - After deselecting the attributes, use the Apply Filters button to generate a working relation that removes those attributes. Notice how the working relation changes, and has fewer attributes than the base relation.
    - First, try using the Apriori algorithm with the default parameters. Record the generated rules.
    - Vary the number of rules generated (click on the command that you are running). Try 20, 30, ... Record how many rules you have to generate before generating a rule containing type=mammal.

- Vary the maximum support until a rule containing type=mammal is the top rule generated. Record the maximum support needed.
- Select one generated rule that was interesting to you. Why was it interesting? What does it mean? Check its confidence and support – are they high enough?
- Suggest one improvement to the Apriori implementation in WEKA that would have made this data mining lab easier to accomplish.

14. Demonstrate to predict the Numerical Values in the given Data Set is using Regression Methods.

Session-7

15. Demonstrate the classification rule process on the student.arff, employee.arff and labor.arff datasets using the following algorithms:
    a. Logistic Regression
    b. Decision Tree
    c. Naïve Bayes
16. Demonstrate the classification rule process on the student.arff, employee.arff and labor.arff datasets using the following algorithms:
    a. K-Nearest Neighbour
    b. SVM

Session-8

17. Perform the following:
    - Demonstrate performing classification on various data sets.
    - Load each dataset into WEKA and run ID3, J48 classification algorithm.
    - Study the classifier output. Compute entropy values, Kappa statistic.
    - Extract if-then rules from the decision tree generated by the classifier.
    - Observe the confusion matrix.
18. Perform the following:
    - Load each dataset into WEKA and perform Naïve-bayes classification and k-Nearest Neighbour classification.
    - Interpret the results obtained.
    - Plot RoC Curves

- Compare classification results of ID3, J48, Naïve-Bayes and k-NN classifiers for each dataset, and deduce which classifier is performing best and poor for each dataset and justify.

Session-9

19. Demonstrate Clustering features in Large Databases with noise.
20. Implement simple K-Means Algorithm to demonstrate the clustering rule on the following datasets:
    a. iris.arff
    b. student.arff
21. Perform the following:
    - Load each dataset into WEKA and run simple k-means clustering algorithm with different values of k (number of desired clusters).
    - Study the clusters formed.
    - Observe the sum of squared errors and centroids, and derive insights.
    - Explore other clustering techniques available in WEKA.
    - Explore visualization features of WEKA to visualize the clusters.
    - Derive interesting insights and explain.

Session-10

22. Implement Hierarchical Clustering Algorithm to demonstrate the clustering rule process in the following datasets;
    a. employee.arff
    b. student.arff
23. Implement Density based Clustering Algorithm to demonstrate the clustering rule process on dataset employee.arff.

2.14 SUMMARY

Data mining (also known as knowledge discovery from databases) is the process of extraction of hidden, previously unknown and potentially useful information from databases. The outcome of the extracted data can be analyzed for the future planning and development perspectives.

Data mining steps in the knowledge discovery process are as follows:

- **Data Cleaning**- The removal of noise and inconsistent data.
- **Data Integration** - The combination of multiple sources of data.

- **Data Selection** - The data relevant for analysis is retrieved from the database.
- **Data Transformation** - The consolidation and transformation of data into forms appropriate for mining.
- **Data Mining** - The use of intelligent methods to extract patterns from data.
- **Pattern Evaluation** - Identification of patterns that are interesting.
- **Knowledge Presentation** - Visualization and knowledge representation techniques are used to present the extracted or mined knowledge to the end user

In this lab course you were given exposure to work with WEKA. WEKA is a collection of machine learning algorithms for data mining tasks. It contains tools for data preparation, classification, regression, clustering, association rules mining, and visualization.Found only on the islands of New Zealand, the WEKA. WEKA is open source software issued under the GNU General Public License. The video links for the courses are available at Online Lab Resources.

2.15 FURTHER READINGS

1. Eibe Frank, Mark A. Hall, and Ian H. Witten (2016). The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, Fourth Edition, 2016.
2. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten (2009). The WEKA Data Mining Software: An Update. SIGKDD Explorations, Volume 11, Issue 1.
3. Jason Bell (2020) Machine Learning: Hands-On for Developers and Technical Professionals, Second Edition, Wiley.
4. Richard J. Roiger (2020) Just Enough R! An Interactive Approach to Machine Learning and Analytics, CRC Press.
5. Parteek Bhatia (2019) Data Mining and Data Warehousing Principles and Practical Techniques, Cambridge University Press.
6. Mark Wickham (2018) Practical Java Machine Learning Projects with Google Cloud Platform and Amazon Web Services, APress.
7. AshishSingh Bhatia, Bostjan Kaluza (2018) Machine Learning in Java - Second Edition, Packt Publishing.
8. Richard J. Roiger (2016) Data Mining: A Tutorial-Based Primer, CRC Press.
9. Mei Yu Yuan (2016) Data Mining and Machine Learning: WEKA Technology and Practice, Tsinghua University Press (in Chinese).
10. Jürgen Cleve, Uwe Lämmel (2016) Data Mining, De Gruyter (in German).
11. Eric Rochester (2015) Clojure Data Analysis Cookbook - Second Edition, Packt Publishing.
12. Boštjan Kaluža (2013) Instant WEKA How-to, Packt Publishing.
13. Hongbo Du (2010) Data Mining Techniques and Applications, Cengage Learning.

2.16 WEBSITE REFERENCES

1. https://www.cs.waikato.ac.nz/ml/weka
2. https://weka.wikispaces.com

2.17 ONLINE LAB RESOURCES

1. Weka have put together several free online courses available at https://www.cs.waikato.ac.nz/ml/weka/courses.html that teach machine learning and data mining using Weka.
2. The videos for the courses are available on Youtube available at URL https://www.youtube.com/user/WekaMOOC .
3. FAQs on WEKA: https://waikato.github.io/weka-wiki/faq/
4. WEKA Downloads: https://waikato.github.io/weka-wiki/downloading_weka/
5. DATASETS: https://waikato.github.io/weka-wiki/datasets/