

Handout

for

One day Workshop with Hands-on Training
on
“Machine Learning and Its Applications”



Prepared and Presented
By

Dr. D. Asir Antony Gnana Singh, B.E., M.E., M.B.A., Ph.D..
Department of CSE, Anna University,
BIT Campus, Tiruchirappalli-620 0024

at
UCE, Anna University, BIT-Campus,
Tiruchirappalli-620 0024
on
28th September 2019

Machine Learning Codes Using Java

Contents

Chapter 1 Software Installation and Configuration

- Part I Downloading and Installing NetBeans IDE 8.2
- Part II Downloading and Installing Weka 3.8.3
- Part III Creating a new Java project space in NetBeans IDE 8.2 to develop Machine learning models

Chapter 2 Dataset Preparation

- Part I Preparation of Dataset

Chapter 3 Developing Ml models and performance evaluation

- Part I Developing Java code to construct and evaluate the Machine learning models
- Part II Additional Workouts

Chapter 4 Prediction using Ml model

- Part I Preparing the unknown (unlabeled) dataset
- Part II Developing the Ml Models and Perform Prediction
- Part III Additional Workouts

Chapter 3 Performing feature selection

- Part I Selecting important features from dataset
- Part II Additional Workouts

Chapter 1

Software Installation and Configuration

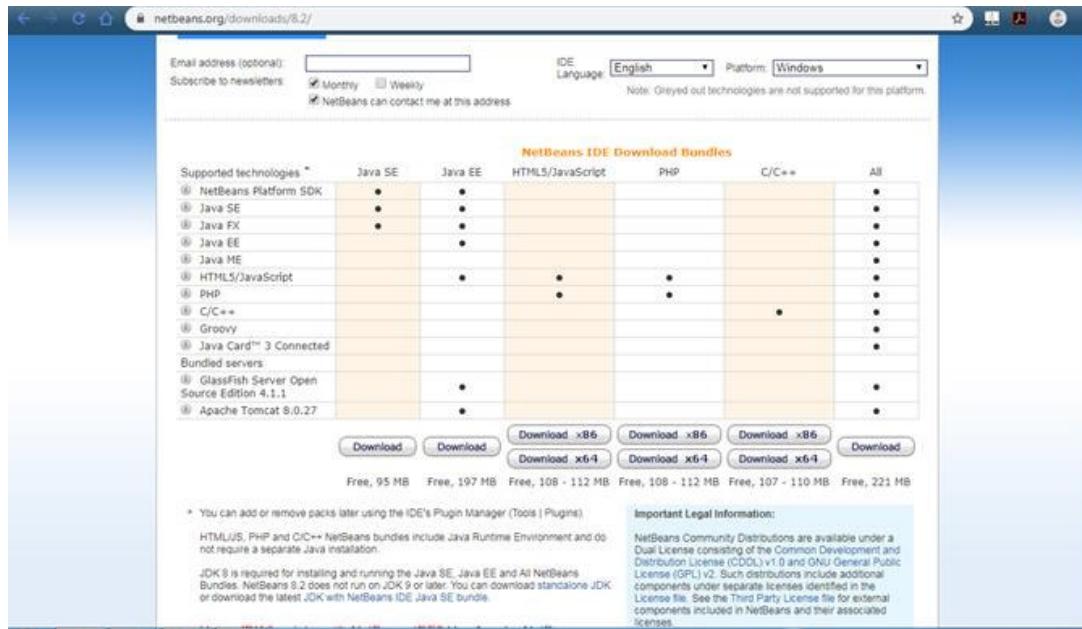
Part I

Downloading and Installing NetBeans IDE 8.2

Step I: Download the NetBeans IDE 8.2

1. Go to the official website of NetBeans IDE 8.2 using the following link to download:

<https://netbeans.org/downloads/8.2/>

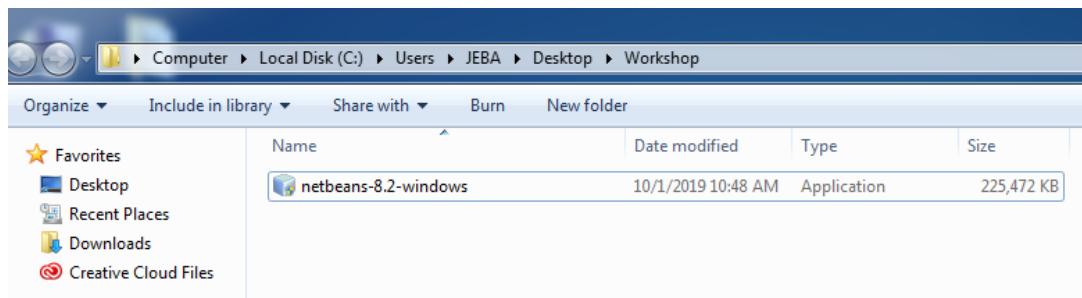


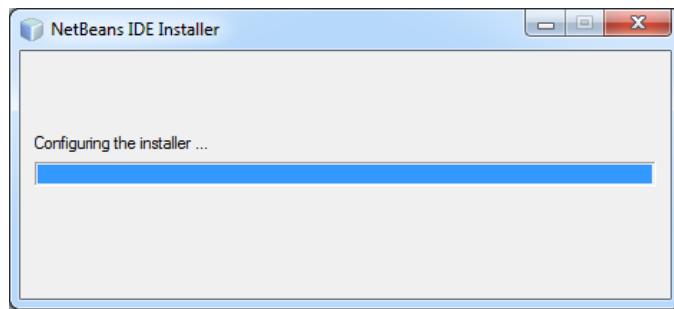
Snapshot of web page (<https://netbeans.org/downloads/8.2/>)

2. Choose the option “English” for IDE Language: and choose the option “Windows” option for Platform:
3. Click the download button provided for the option “All” for the NetBeans IDE Download Bundles to download the IDE.
4. (netbeans-8.2-windows)

Step II: Install the IDE

Double-click the installer file (netbeans-8.2-windows.exe) to run it.

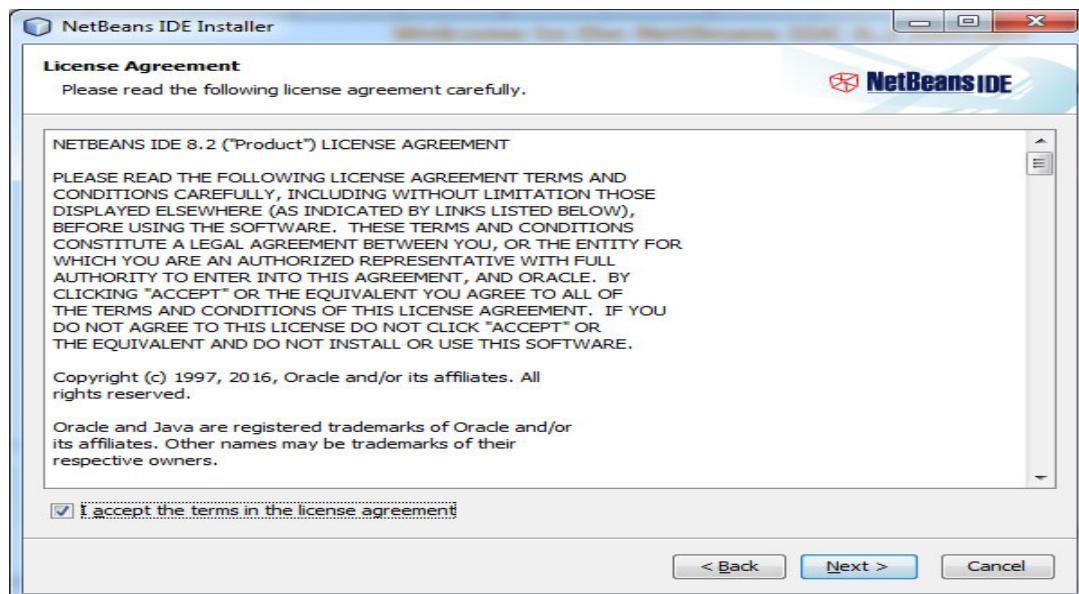




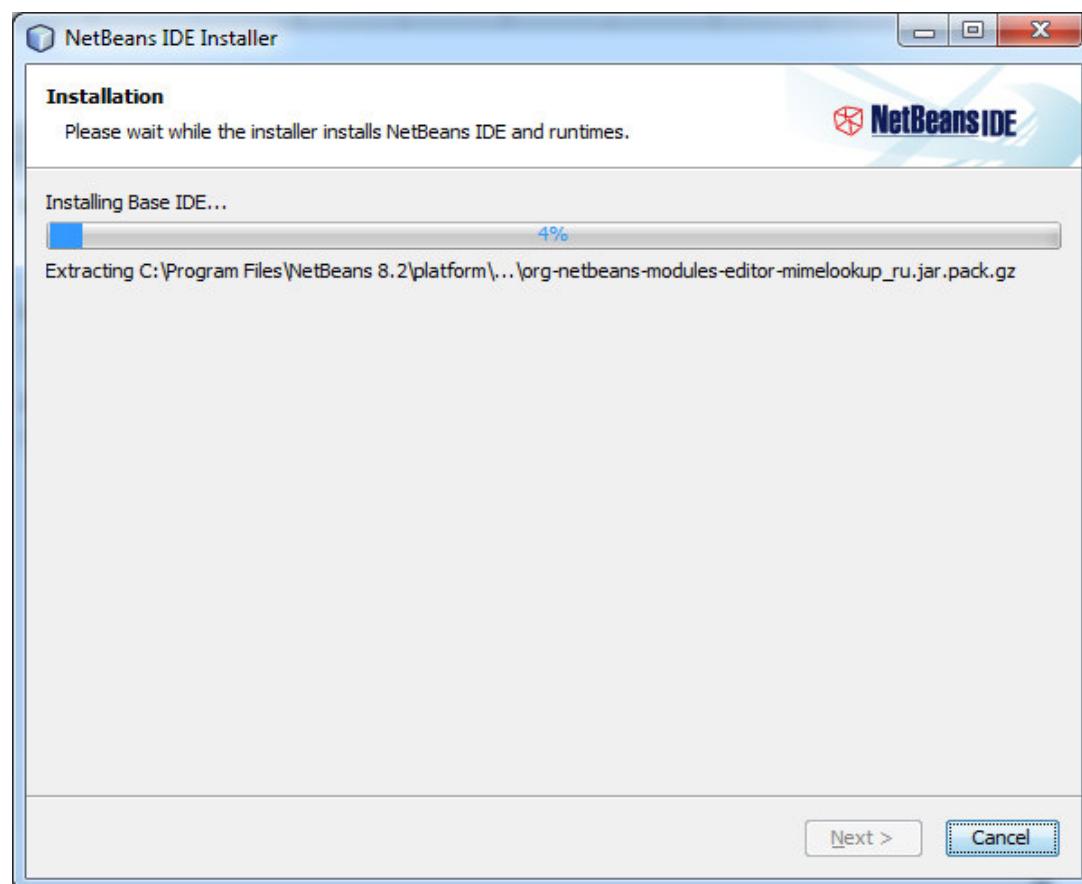
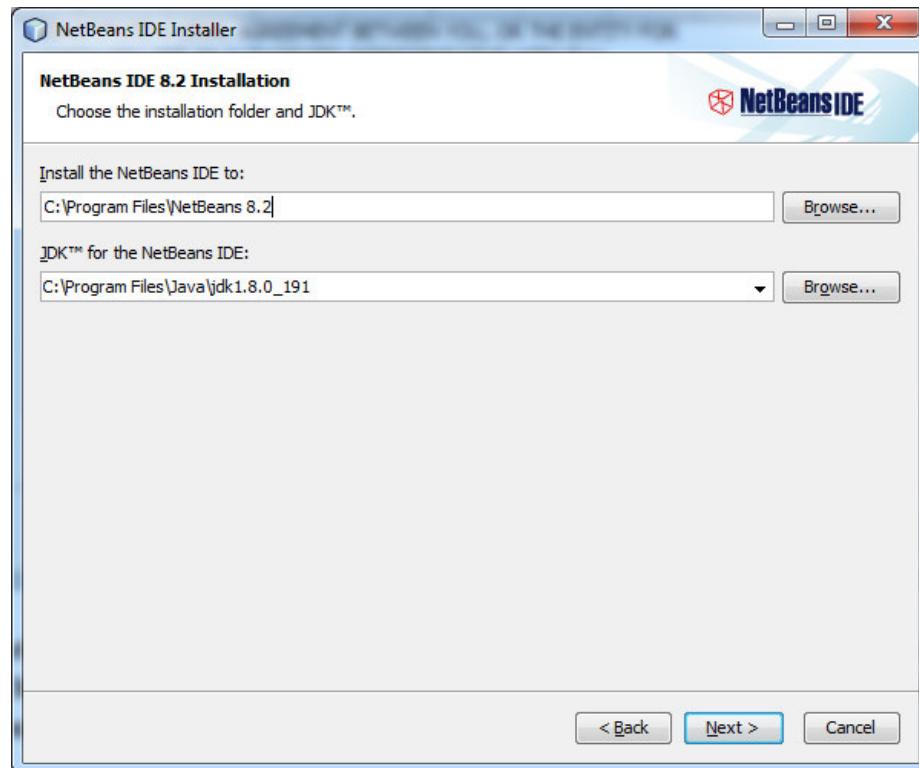
1. Click the Next button to continue



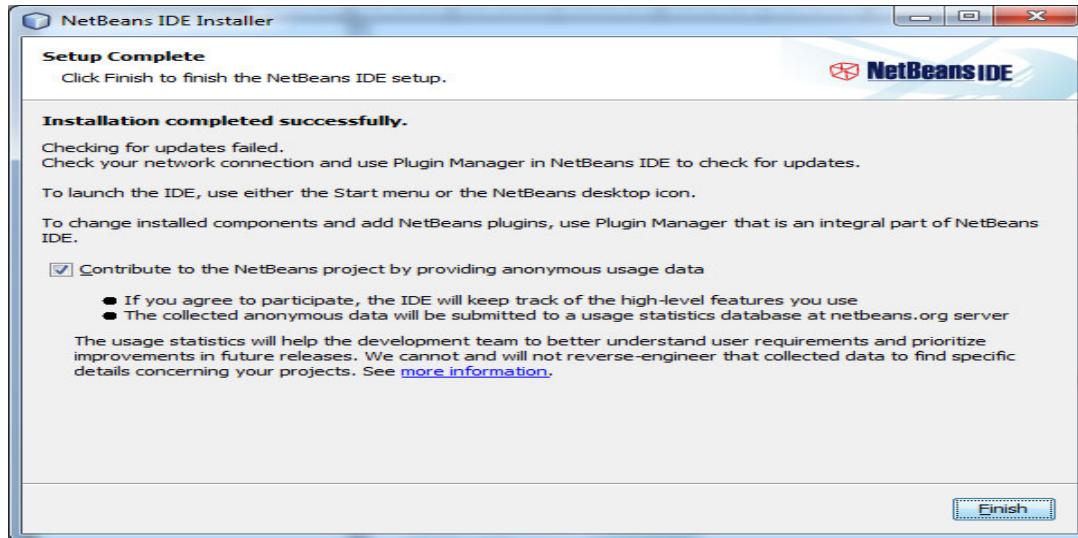
2. Click the button I Agree (That means you accept the terms of agreement) to continue



3. Choose the installation folder to install the IDE 8.2 and click next button to continue

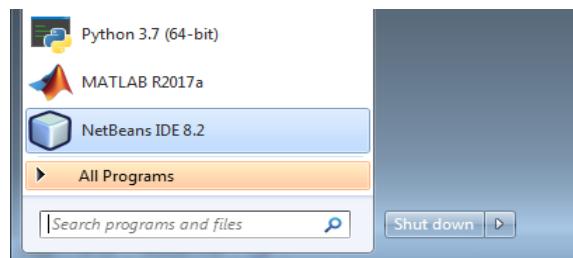


- After Installation complete the finish button to complet the installation

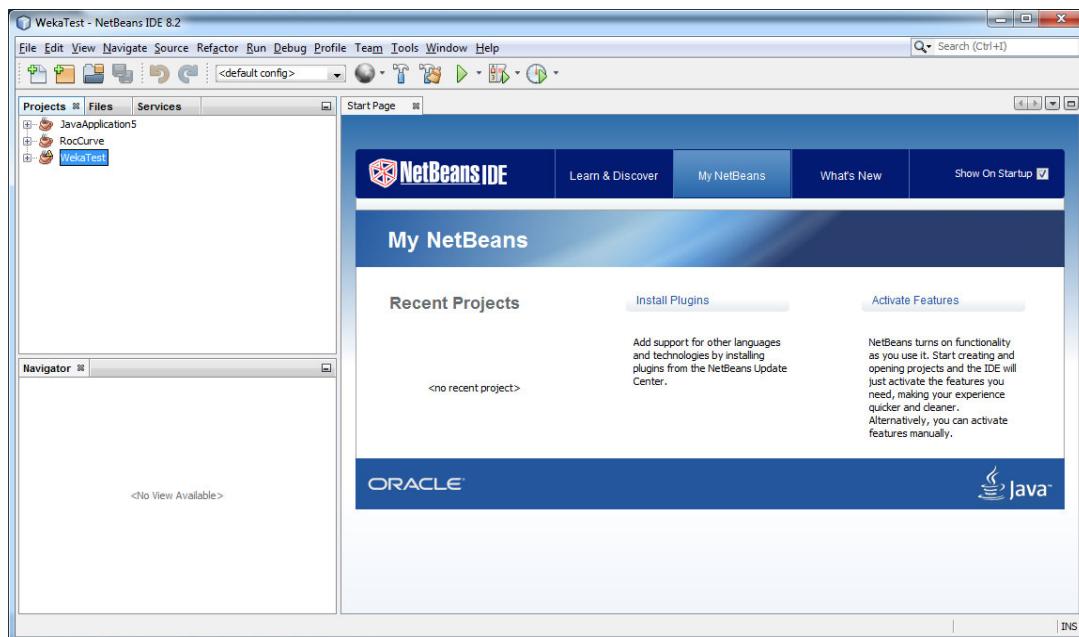


Step III: Explore the IDE

- Double-click the icon of the IDE to open the IDE.



- The IDE will appear as shown in the following screenshot :



Part II

Downloading and Installing Weka 3.8.3

Step I: Download the Weka 3.8.3

1. Go to the official website of Weka using the following link for downloading:

<https://www.cs.waikato.ac.nz/ml/weka/downloading.html>

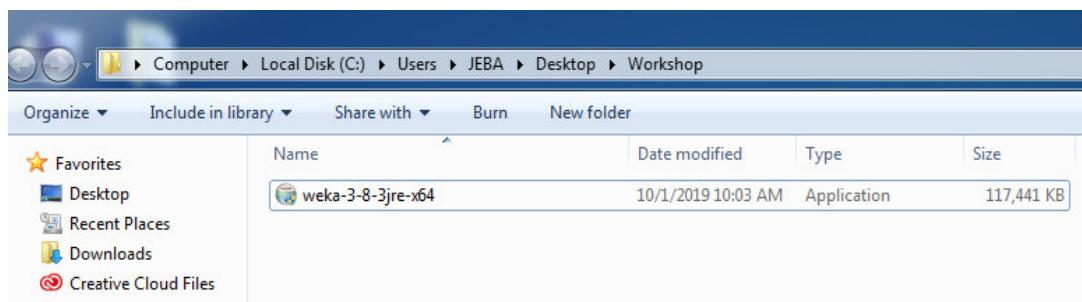
The screenshot shows a web browser window with the URL <https://www.cs.waikato.ac.nz/ml/weka/downloading.html>. The page has a header with tabs: Project, Software (which is underlined), Book, Courses, Publications, People, and Related. The main content area is titled "Downloading and installing Weka". It discusses two versions: Weka 3.8 (stable) and Weka 3.9 (development). It notes that Weka 3.8 receives bug fixes only, while 3.9 receives new features. It mentions a package management system for adding functionality. A note for high-pixel-density Windows users is present. Below this, a section for the "Stable version" is shown, specifically for Weka 3.8. It lists a "Windows" option with a download link: "Click here to download a self-extracting executable for 64-bit Windows that includes Oracle's 64-bit Java VM 1.8 (weka-3-8-3jre-x64.exe; 120.3 MB)".

Snapshot of Web page (<https://www.cs.waikato.ac.nz/ml/weka/downloading.html>)

2. Click the down load link of (Click here to Down load the download a self-extracting executable for 64-bit Windows that includes Oracle's 64-bit Java VM 1.8) from the Window section of Stable version to download the weka-3-8-3jre-x64.exe. (120.3 MB)

Step II: Install the Weka 3.8.3

1. Double-click the installer file (weka-3-8-3jre-x64.exe.) to run it.



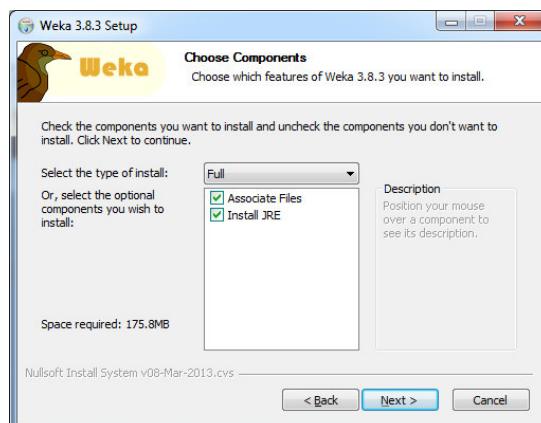
2. Click the Next button to continue



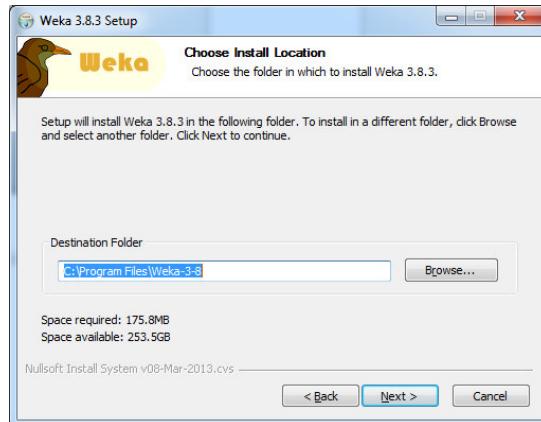
3. Click the button I Agree (That means you accept the terms of agreement) to continue



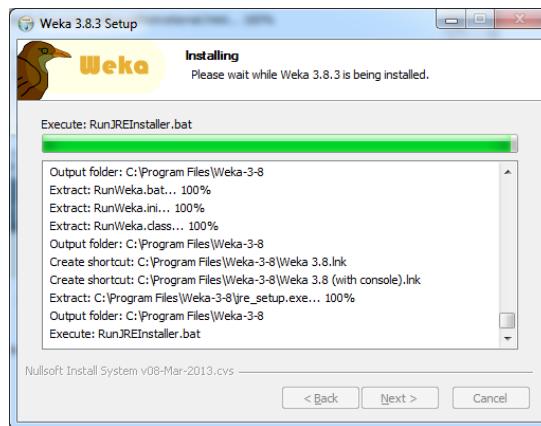
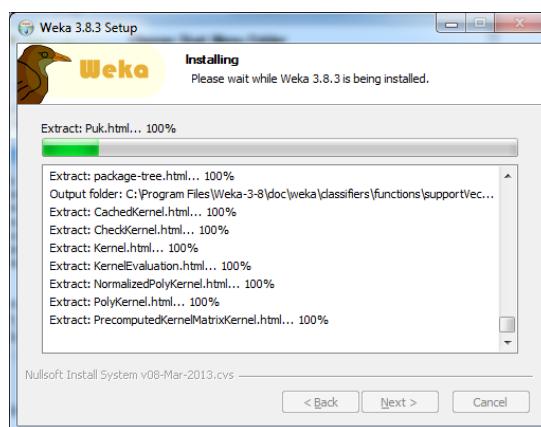
4. Select the option “Full” from the dropdown menu “Select the type of Installation:”
Click on the checkboxes “Associated Files” and “Install JRE” and Click the next button to continue



5. Choose the directory to install the Weka 3.8.3 and click next button to continue



6. Click the Install button to install



7. At the end of the installation a “Java Setup-Welcome” popup window will appear to setup the compatible version of Java for Weka 3.8.3. Click the Install button to setup the Java for Weka 3.8.3



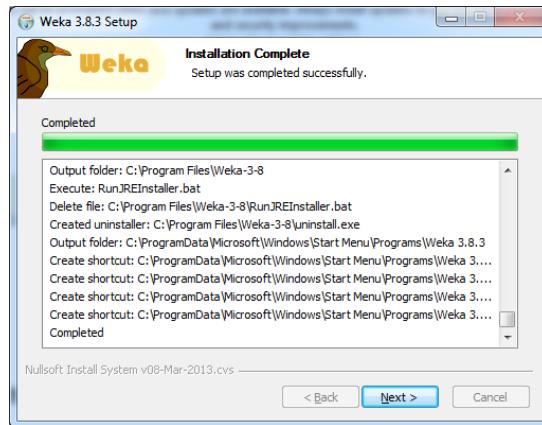
8. Installation of Java is started as following snapshot



9. After completion of the installation of the Java the popup window will appear as follow snapshot and click the close button to close the popup window



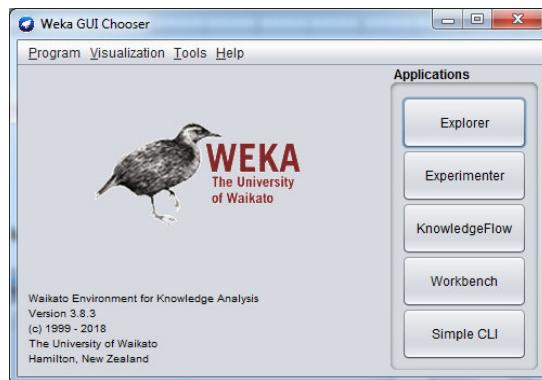
10. Click the next button of the Weka 3.8.3 Setup window to complete the installation



11. Click on the Start Weka checkbox and Click the finish button to launch the Weka GUI Chooser



12. The Weka GUI Chooser will appear as the screenshot given below:



Part III

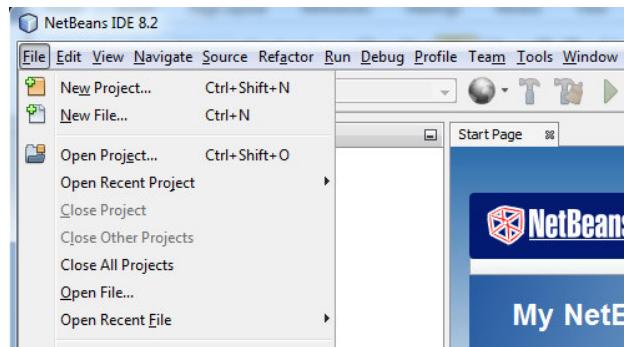
Creating a new Java project space in NetBeans IDE 8.2 to develop Machine learning model

Step I:

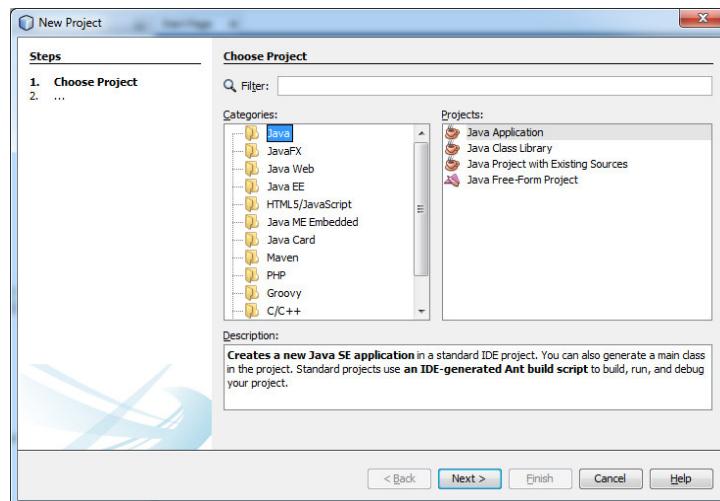
1. Explore the NetBeans IDE 8.2



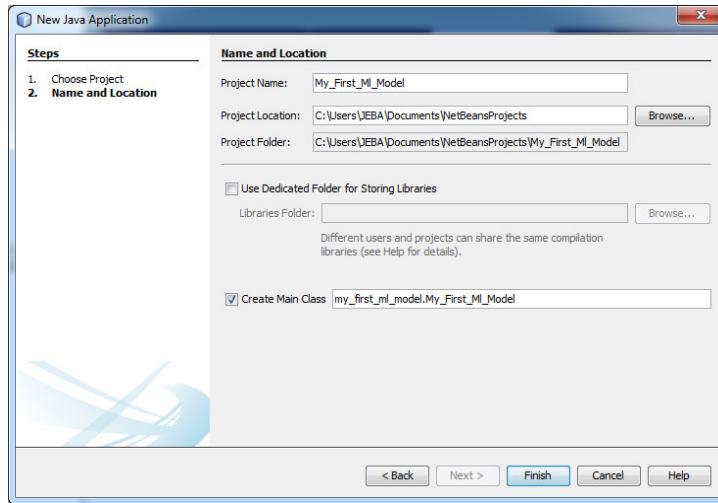
2. Select the File menu (New Project popup window will appear as shown in below screenshot)



3. Choose the "Java menu" from the dropdown menu "Categories" and choose "Java Application" from the dropdown menu "Project" and click on the "New Project" Tab (New Java Application popup window will appear)

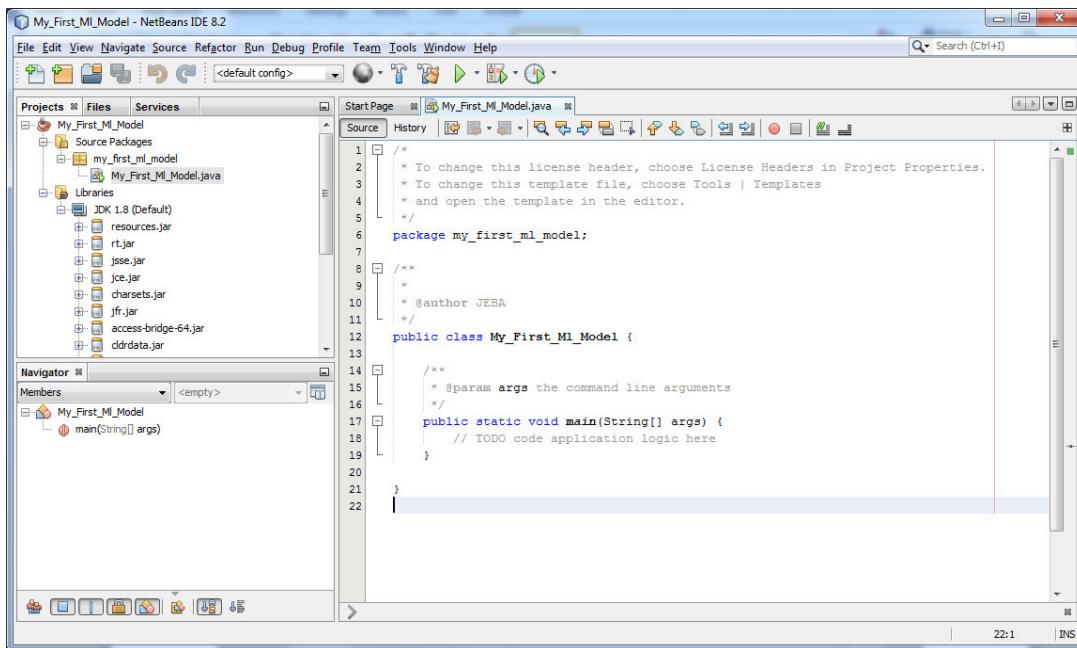


4. Provide the Project Name as “My_First_Ml_Model” and click finish



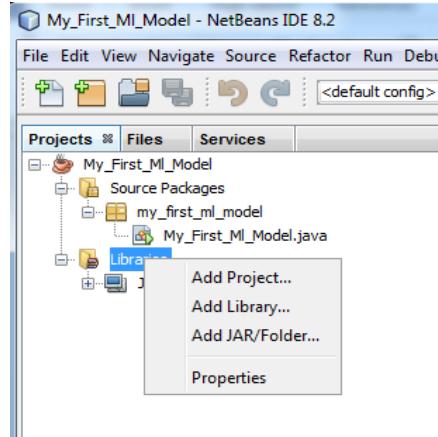
5. Choose the “Projects” under this tab you can see:

- The Java project is created with name of “My_First_Ml_Model” and inside of the project the two directories are created one for Source package and another for Libraries.
- Inside of the source package the main package is named as “My_First_Ml_Model”.
- Inside of the main package a main java class file is created with the file name of “My_First_Ml_Model.java
- Inside of the libraries the JAR (Java ARchive) files for the JDK (Java Development Kit) are available.

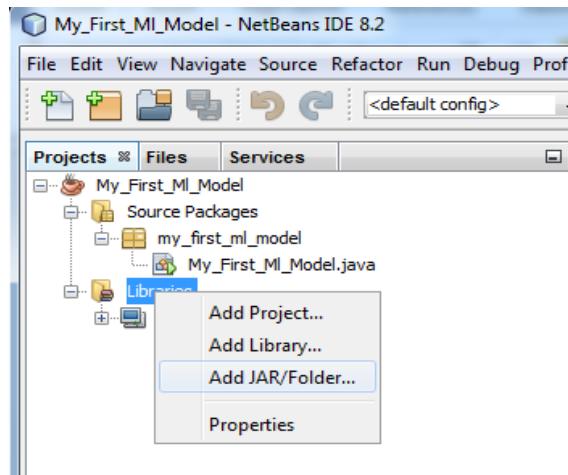


**Step II: (Configure the Weka API 3.8.3 and NetBeans IDE 8.2) Add the Java jar file
into the Project Libraries using the following steps:**

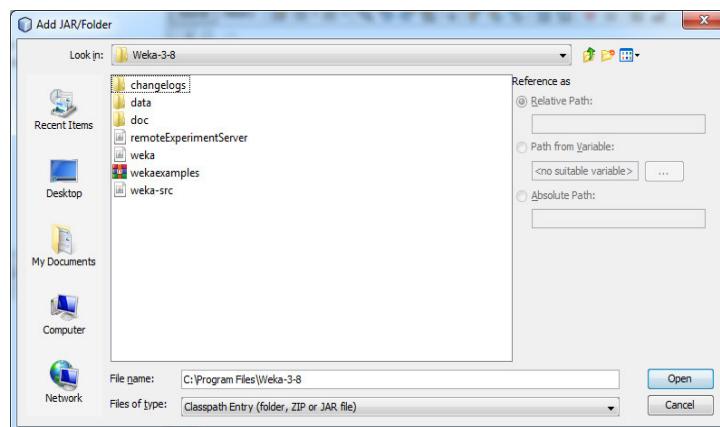
1. Do right-click on the Libraries folder as shown below screenshot



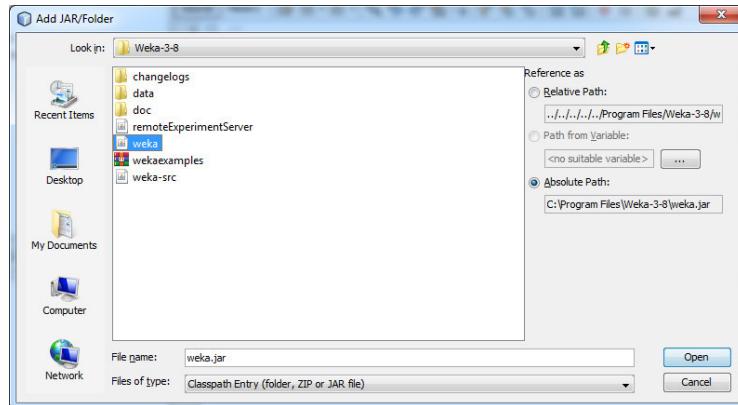
2. Click on the menu “Add JAR/Folder” as shown in the below screenshot



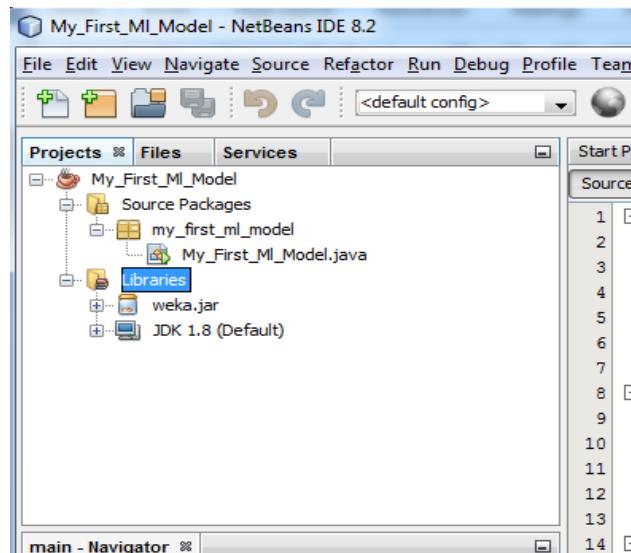
3. Add JAR/Folder popup window will appear and Select the Weka installed folder (The path of where the Weka is installed) (For example: “C:\Program Files\Weka-3-8”) as shown in below screenshot.



4. Click on the Weka jar file from the directory (C:\Program Files\Weka-3-8) as shown in the below screenshot and click on the open button



5. Now the Weka.jar is included in the Libraries as shown in the blow screenshot.



Chapter 2 **Dataset Preparation**

Part I **Preparation of Dataset**

Step I: Prepare the dataset (in ARFF (attribute relation file format))

Dataset consist of M instances with N attributes followed by the class label as shown below:

Attributes					Class/Label
X ₁	X ₂	X ₃	...	X _N	C
X ₁₁	X ₂₁	X ₃₁	...	X _{N1}	c ₁
X ₁₂	X ₂₂	X ₃₂	...	X _{N2}	c ₂
X ₁₃	X ₂₃	X ₃₃	...	X _{N3}	c ₁
.
.
X _{1M}	X _{2M}	X _{3M}	...	X _{NM}	c ₂

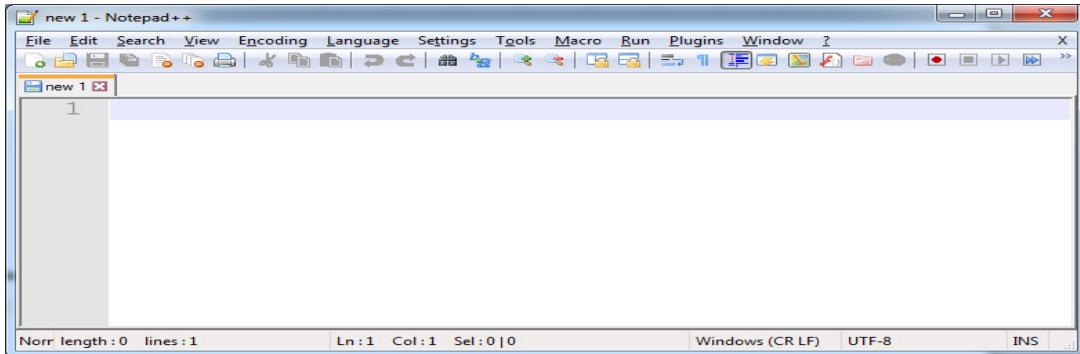
Representation of Dataset

Note: The step by step procedure is given to prepare medical dataset entitled diabetes dataset. Dataset contains eight attribute and one class for 768 instances. This dataset is used to predict whether a person has diabetes or not.

Description of Attribute and class label

1. Number of times pregnant
1. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
2. Diastolic blood pressure (mm Hg)
3. Triceps skin fold thickness (mm)
4. 2-Hour serum insulin (mu U/ml)
5. Body mass index (weight in kg/(height in m)²)
6. Diabetes pedigree function
7. Age (years)
8. Class variable (0 or 1)

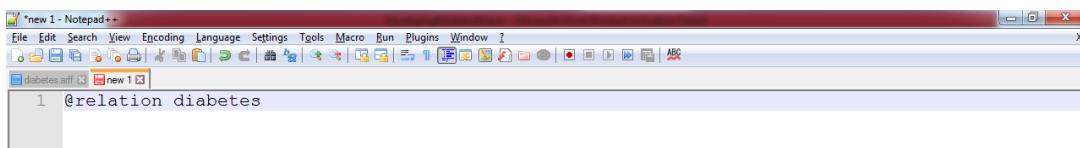
1. Open Notepad/Notepad++ and



2. Define the name of the dataset:

Syntax: @ relation <dataset_name>

Code: @relation diabetes



3. Define the attributes and class label of the dataset:

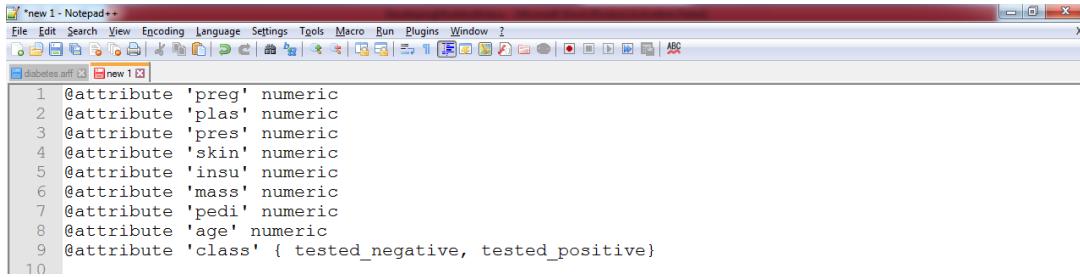
Syntax: @ attribute <name of the attribute > <attribute type>

Syntax: @ attribute <class label name > <attribute type>

Code: @attribute 'preg' numeric

```
@attribute 'plas' numeric  
@attribute 'pres' numeric  
@attribute 'skin' numeric  
@attribute 'insu' numeric  
@attribute 'mass' numeric  
@attribute 'pedi' numeric  
@attribute 'age' numeric
```

Code: @attribute 'class' { tested_negative, tested_positive}



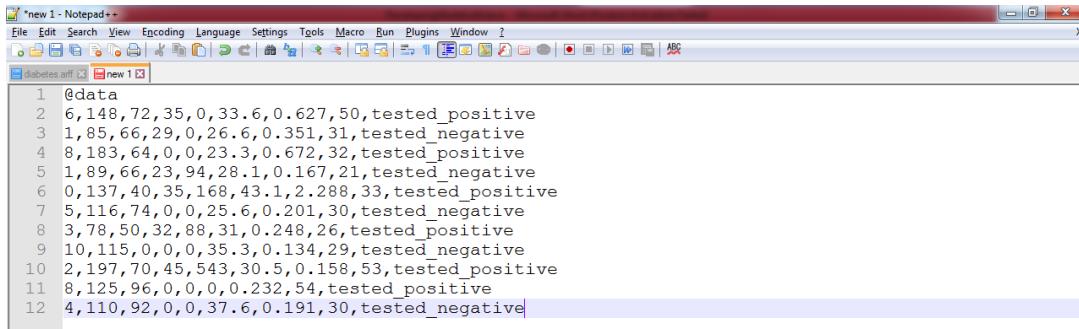
4. Define the instances of the dataset:

Syntax: @data

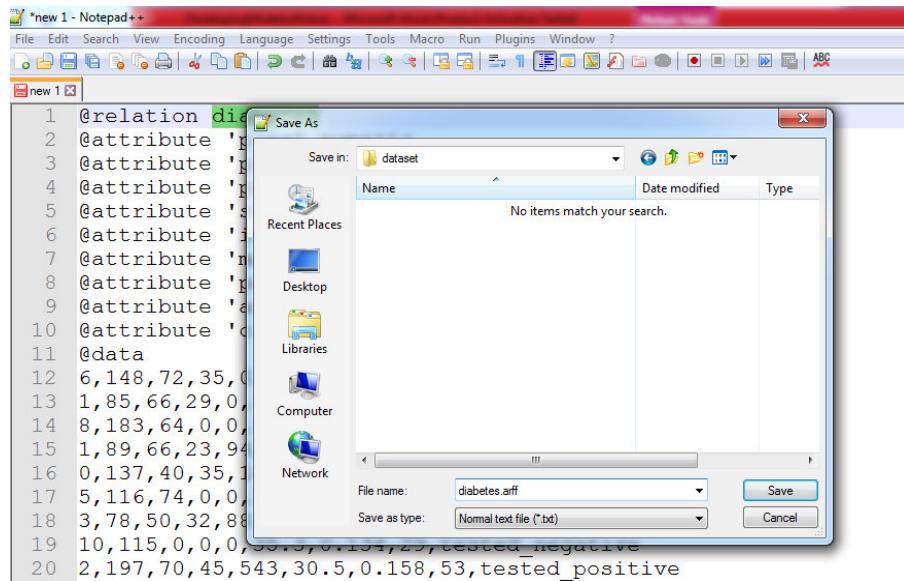
< value for attribute 1, value for attribute 2, value for attribute n>

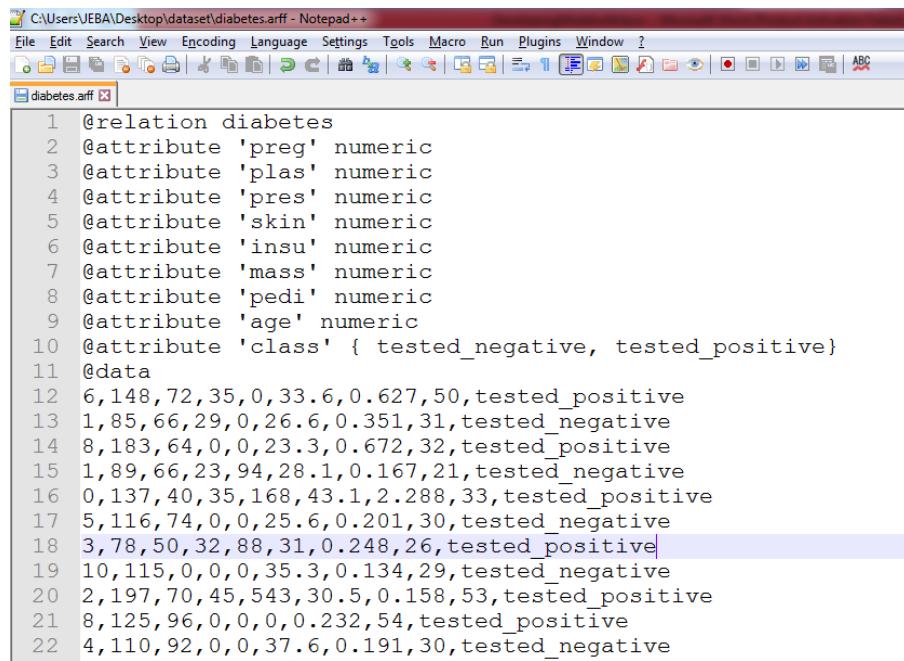
Code:

```
@data
6,148,72,35,0,33.6,0.627,50,tested_positive
1,85,66,29,0,26.6,0.351,31,tested_negative
8,183,64,0,0,23.3,0.672,32,tested_positive
1,89,66,23,94,28.1,0.167,21,tested_negative
0,137,40,35,168,43.1,2.288,33,tested_positive
5,116,74,0,0,25.6,0.201,30,tested_negative
3,78,50,32,88,31,0.248,26,tested_positive
10,115,0,0,0,35.3,0.134,29,tested_negative
2,197,70,45,543,30.5,0.158,53,tested_positive
8,125,96,0,0,0,0.232,54,tested_positive
4,110,92,0,0,37.6,0.191,30,tested_negative
10,168,74,0,0,38,0.537,34,tested_positive
```



5. Save the file with the file extension of arff shown in screenshots below:





The screenshot shows the Notepad++ application window with the file "diabetes.arff" open. The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Tools, Macro, Run, Plugins, Window, and Help. The toolbar contains various icons for file operations like Open, Save, Find, and Print. The code editor displays the ARFF file format, starting with a header section defining attributes and ending with a data section containing 22 entries. The 18th entry is highlighted with a light blue background.

```
1 @relation diabetes
2 @attribute 'preg' numeric
3 @attribute 'plas' numeric
4 @attribute 'pres' numeric
5 @attribute 'skin' numeric
6 @attribute 'insu' numeric
7 @attribute 'mass' numeric
8 @attribute 'pedi' numeric
9 @attribute 'age' numeric
10 @attribute 'class' { tested_negative, tested_positive}
11 @data
12 6,148,72,35,0,33.6,0.627,50,tested_positive
13 1,85,66,29,0,26.6,0.351,31,tested_negative
14 8,183,64,0,0,23.3,0.672,32,tested_positive
15 1,89,66,23,94,28.1,0.167,21,tested_negative
16 0,137,40,35,168,43.1,2.288,33,tested_positive
17 5,116,74,0,0,25.6,0.201,30,tested_negative
18 3,78,50,32,88,31,0.248,26,tested_positive
19 10,115,0,0,0,35.3,0.134,29,tested_negative
20 2,197,70,45,543,30.5,0.158,53,tested_positive
21 8,125,96,0,0,0,0.232,54,tested_positive
22 4,110,92,0,0,37.6,0.191,30,tested_negative
```

Chapter 3
Developing ML models and performance
evaluation

Part I

Developing Java Code to construct and evaluate the Machine learning models

Outline of the program

The steps to developed the java program is shown in the below Figure. Initially, the required packages are imported. The dataset is loaded to develop the machine learning mode. Machine learning model is evaluated based any one of the criteria such a percentage split, n fold cross validation method, and training set as the test set. The evaluated results are displayed in terms of performance evaluation metrics such as accuracy, TP Rate, FP Rate, Precision, Recall, F-Measure and ROC Area.

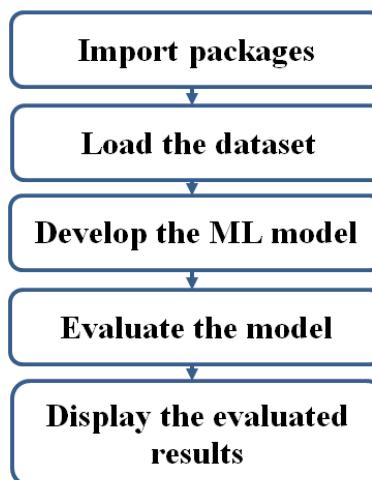
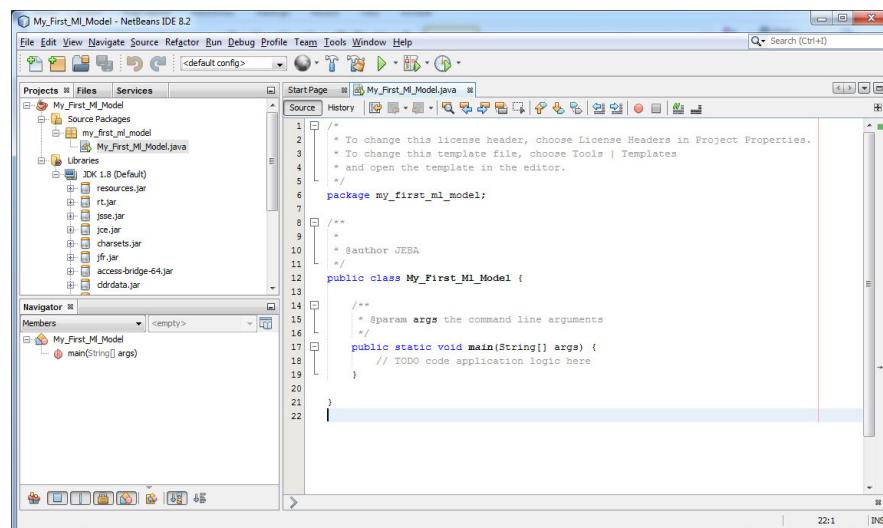


Figure: Steps to develop a Java code for construct and evaluate the MI models

Note: Open the Code editing space of the main class file “My_First_Ml_Model.java” (as shown in below screenshot) to write the code for developing machine learning models



Step I: Develop the Java code to construct and evaluate the Machine learning models

1. Import the required packages:

a. Import Package to develop the MI model (J48)

Definition:	J48 class is used to generate decision tree-based machine learning model.
Code:	import weka.classifiers.trees.J48;
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html

b. Import package to access the dataset from a file

Definition:	DataSource class is used loading data from files
Code:	Import weka.core.converters.ConverterUtils.DataSource;
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/core/converters/ConverterUtils.DataSource.html

c. Import package to evaluating the performance of the MI model

Definition:	Evaluation class for evaluating machine learning models
Code:	import weka.classifiers.Evaluation;
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

d. Import package to generate the random number

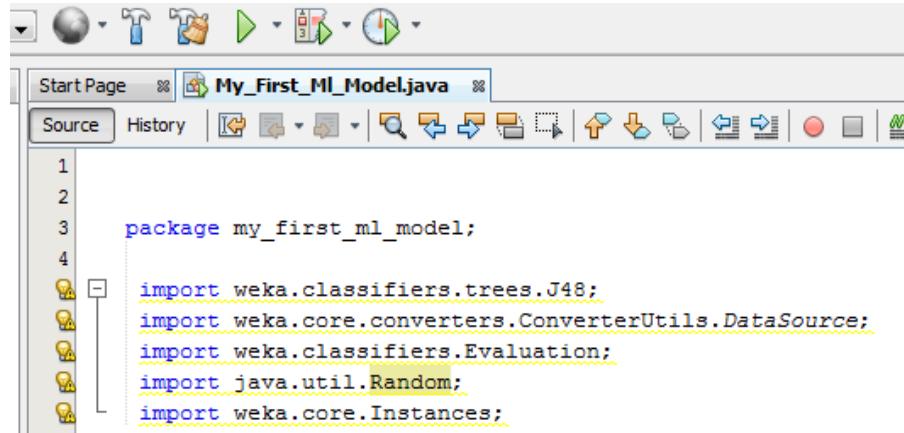
Definition:	Random class is used to generate a stream of pseudorandom numbers
Code:	import java.util.Random;
Ref. URL:	https://docs.oracle.com/javase/8/docs/api/java/util/Random.html

e. Import package to handle the instance of the dataset

Definition:	Instances class is used to handle the ordered set of weighted instances.
Code:	import weka.core.Instances;
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/core/Instances.html

Snapshot of code to import required classes:

```
import weka.classifiers.trees.J48;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.Evaluation;
import java.util.Random;
import weka.core.Instances;
```



```
1
2
3 package my_first_ml_model;
4
5 import weka.classifiers.trees.J48;
6 import weka.core.converters.ConverterUtils.DataSource;
7 import weka.classifiers.Evaluation;
8 import java.util.Random;
9 import weka.core Instances;
```

Step II: Define the main class and the main method

Note: Below a Java code is provided for displaying the string "HelloWorld"

```
public class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the
string.
    }
}
```

- public class: Any other class can access a public field or method.
- public method: This method is public and therefore available to anyone.
- static method: This method can be run without having to create an instance of the class MyClass.
- void method: This method does not return anything.
- (String[] args): This method takes a String argument. Note that the argument args can be anything — it's common to use "args" but we could instead call it "stringArray".

Ref. URL: <https://docs.oracle.com/javase/tutorial/getStarted/application/index.html>

Definition:	<p>public class: Any other class can access a public field or method. public method: This method is public and therefore available to anyone. static method: This method can be run without having to create an instance of the class MyClass. void method: This method does not return anything. (String[] args): This method takes a String argument. Note that the argument args can be anything — it's common to use "args" but we could instead call it "stringArray". The throws keyword indicates what exception type may be thrown by a method.</p>
Code:	<pre>public class My_First_Ml_Model { public static void main(String[] args) throws Exception {</pre>
Ref. URL:	https://docs.oracle.com/javase/tutorial/getStarted/application/index.html

Snapshot of code to define the Main class and the Main method

```
public class My_First_Ml_Model {  
    public static void main(String[] args) throws Exception {
```

```
My_First_Ml_Model.java  
History | Back | Forward | Stop | Refresh | Save | Open | Close | Minimize | Maximize | Restore | Full Screen | Exit  
import weka.classifiers.trees.J48;  
import weka.core.converters.ConverterUtils.DataSource;  
import weka.classifiers.Evaluation;  
import java.util.Random;  
import weka.core Instances;  
  
public class My_First_Ml_Model {  
    public static void main(String[] args) throws Exception {
```

Step III: Load the dataset from the file

Definition:	DataSource class is used to loading data from files getDataSet() method returns the full dataset, can be null in case of an error. In this code, an object source is created for the class DataSource and the object source is used to invoke the method getDataSet() using dot operator Note: Provide the entire path of the dataset which should be loaded
Code:	DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff"); Instances data = source.getDataSet();
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/core/converters/ConverterUtils.DataSource.html

Snapshot of code to load the full dataset

```
DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");  
Instances data = source.getDataSet();
```

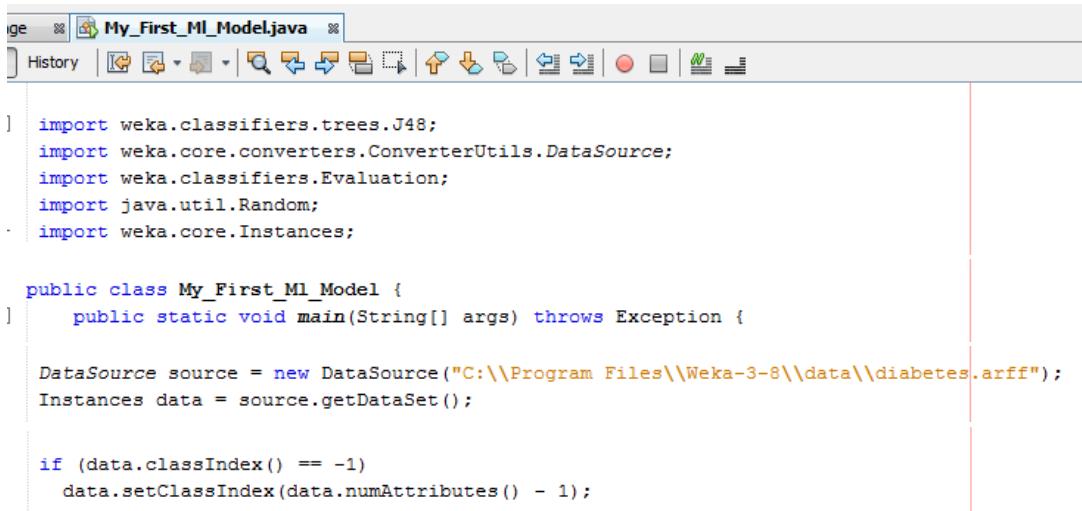
```
My_First_Ml_Model.java  
History | Back | Forward | Stop | Refresh | Save | Open | Close | Minimize | Maximize | Restore | Full Screen | Exit  
import weka.classifiers.trees.J48;  
import weka.core.converters.ConverterUtils.DataSource;  
import weka.classifiers.Evaluation;  
import java.util.Random;  
import weka.core Instances;  
  
public class My_First_Ml_Model {  
    public static void main(String[] args) throws Exception {  
  
        DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");  
        Instances data = source.getDataSet();
```

Step IV: Set the class attribute for the dataset

Definition:	classIndex() Returns the class attribute's index (default it is set -1). setClassIndex(int classIndex) Sets the class index of the dataset. numAttributes() Returns the total number of attributes of the dataset Note: all the above methods are invoked by the object data which is created for the class Instances.
Code:	if (data.classIndex() == -1) data.setClassIndex(data.numAttributes() - 1);
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/core/converter/ConverterUtils.DataSource.html

Snapshot of code to set the class attribute for the dataset

```
if (data.classIndex() == -1)
    data.setClassIndex(data.numAttributes() - 1);
```



The screenshot shows a Java code editor window titled "My_First_Ml_Model.java". The code is as follows:

```
import weka.classifiers.trees.J48;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.Evaluation;
import java.util.Random;
import weka.core.Instances;

public class My_First_Ml_Model {
    public static void main(String[] args) throws Exception {
        DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
        Instances data = source.getDataSet();

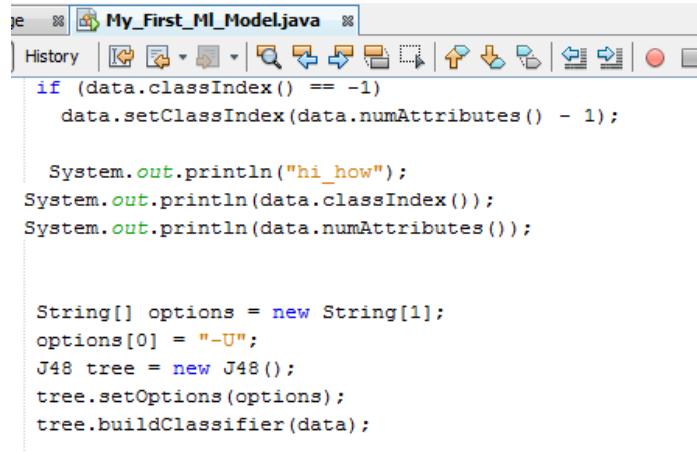
        if (data.classIndex() == -1)
            data.setClassIndex(data.numAttributes() - 1);
    }
}
```

Step V: Develop the machine learning model from the dataset

Definition:	String class represents character strings setOptions method sets the options (Option -U for unpruned tree) for J48 class J48 class for developing a pruned or unpruned C4.5 decision tree buildClassifier method generates the classifier (Machine learning model) Note: tree is the object of the class J48 and the data is the object of the class DataSource
Code:	String[] options = new String[1]; options[0] = "-U"; J48 tree = new J48(); tree.setOptions(options); tree.buildClassifier(data);
Ref. URL:	https://docs.oracle.com/javase/8/docs/api/java/lang/String.html http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html

Snapshot of code to develop the Machine Learning Model from the dataset

```
String[] options = new String[1];
options[0] = "-U";
J48 tree = new J48();
tree.setOptions(options);
tree.buildClassifier(data);
```



```
if (data.classIndex() == -1)
    data.setClassIndex(data.numAttributes() - 1);

System.out.println("hi_low");
System.out.println(data.classIndex());
System.out.println(data.numAttributes());

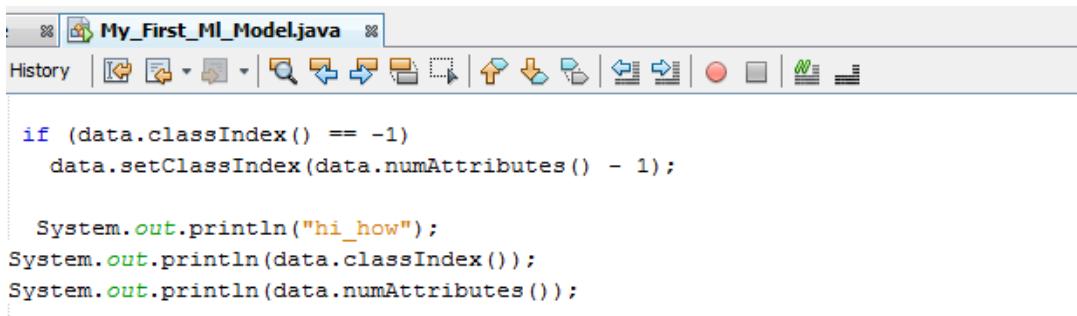
String[] options = new String[1];
options[0] = "-U";
J48 tree = new J48();
tree.setOptions(options);
tree.buildClassifier(data);
```

Step VI: Evaluate the performance of Machine learning model and display the result in console window

Definition:	Evaluation class is used for evaluating machine learning models. crossValidateModel method Performs a (stratified if class is nominal) cross-validation for a classifier on a set of instances. System.out.println(data) is standard output stream that displays value of the data for output toSummaryString method outputs the performance statistics in summary form.
Code:	Evaluation eval = new Evaluation(data); eval.crossValidateModel(tree, data, 10, new Random(1)); System.out.println(eval.toSummaryString("\nResults\n=====\\n", false));
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/classifiers/Evaluation.html https://docs.oracle.com/javase/8/docs/api/java/lang/System.html

Snapshot of code to evaluate the performance of Machine learning model and display the result in console window

```
Evaluation eval = new Evaluation(data);
eval.crossValidateModel(tree, data, 10, new Random(1));
System.out.println(eval.toSummaryString("\nResults\\n=====\\n", false));
```



```
if (data.classIndex() == -1)
    data.setClassIndex(data.numAttributes() - 1);

System.out.println("hi_low");
System.out.println(data.classIndex());
System.out.println(data.numAttributes());
```

```
String[] options = new String[1];
options[0] = "-U";
J48 tree = new J48();
tree.setOptions(options);
tree.buildClassifier(data);

Evaluation eval = new Evaluation(data);
eval.crossValidateModel(tree, data, 10, new Random(1));
System.out.println(eval.toSummaryString("\nResults\n=====\\n", false));
}
```

Note: To complete the program use curly braces at the end of the program one for closing the main class and another one for closing the main method as shown in the below screenshot of complete code.

Snapshot of complete code to develop and evaluate the Machine Learning Model from the dataset

```
import weka.classifiers.trees.J48;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.Evaluation;
import java.util.Random;
import weka.core.Instances;

public class My_First_Ml_Model {
    public static void main(String[] args) throws Exception {

        DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
        Instances data = source.getDataSet();

        if (data.classIndex() == -1)
            data.setClassIndex(data.numAttributes() - 1);

        String[] options = new String[1];
        options[0] = "-U";
        J48 tree = new J48();
        tree.setOptions(options);
        tree.buildClassifier(data);

        Evaluation eval = new Evaluation(data);
        eval.crossValidateModel(tree, data, 10, new Random(1));
        System.out.println(eval.toSummaryString("\nResults\n=====\\n", false));
    }
}

import weka.classifiers.trees.J48;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.Evaluation;
import java.util.Random;
import weka.core.Instances;
public class My_First_Ml_Model {
    public static void main(String[] args) throws Exception {

        DataSource source = new DataSource("C:\\Program Files\\Weka-3-
8\\data\\diabetes.arff");
        Instances data = source.getDataSet();

        if (data.classIndex() == -1)
            data.setClassIndex(data.numAttributes() - 1);
```

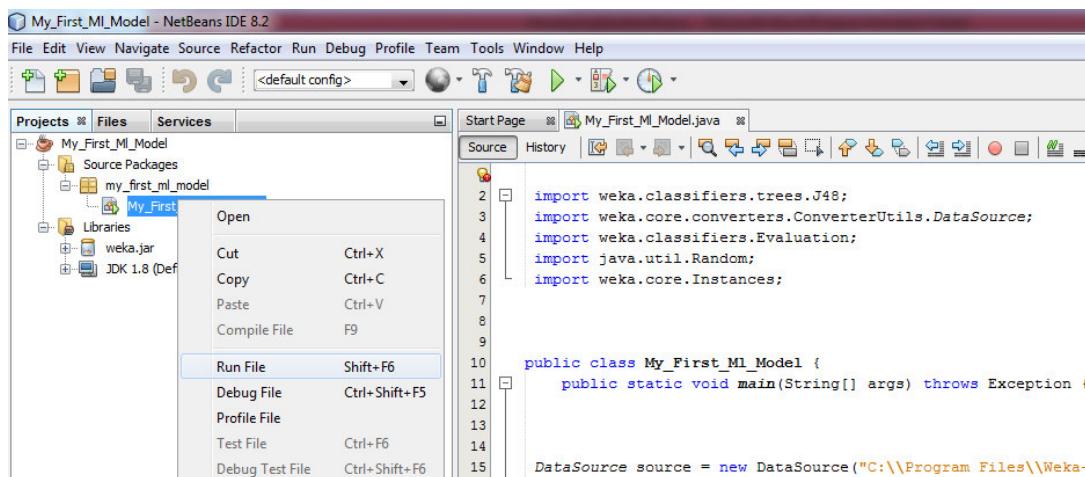
```
String[] options = new String[1];
options[0] = "-U";
J48 tree = new J48();
tree.setOptions(options);
tree.buildClassifier(data);

Evaluation eval = new Evaluation(data);
eval.crossValidateModel(tree, data, 10, new Random(1));
System.out.println(eval.toSummaryString("\nResults\n=====\\n",
false));

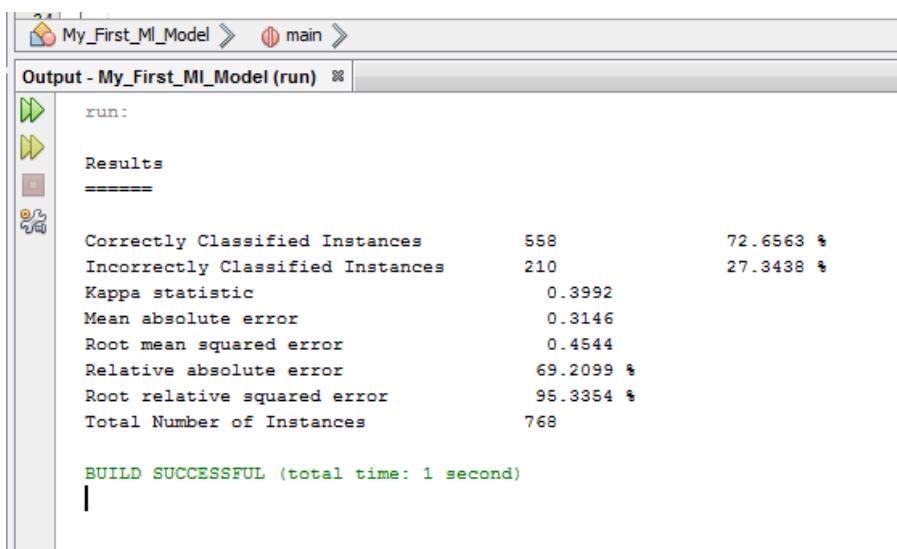
}
```

Step VII: Run the code and display the results

Do right-click on the Java class file named as “My_First_Ml_Model” one drop down menu is appeared and click on the “Run File” option to run the code and display the results as shown in the screenshot below. Or Simply Press the keys (Shift+F6) to run the code display the results.



Note: The results are appeared in the output window as shown in screenshot below



Part II Additional Workouts

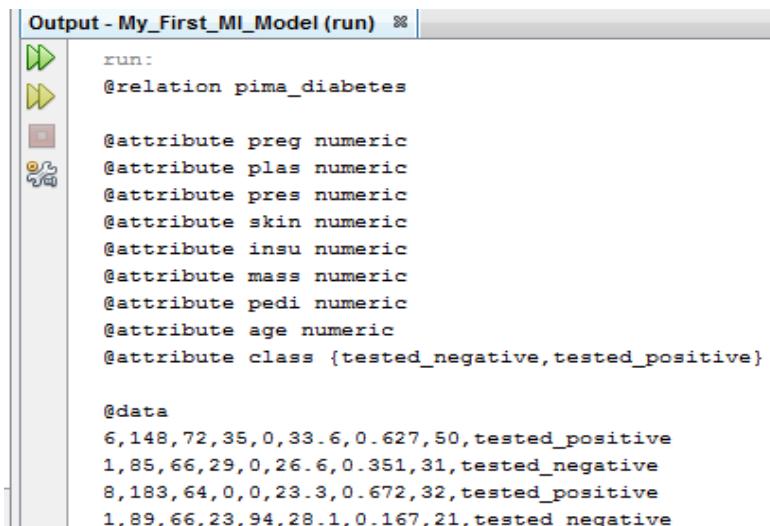
1. Displaying the dataset

a. Display the entire dataset

Screenshot of code:

```
public class My_First_Ml_Model {  
    public static void main(String[] args) throws Exception {  
  
        DataSource source = new DataSource("C:\\\\Program Files\\\\Weka-3-8\\\\data\\\\diabetes.arff");  
        Instances data = source.getDataSet();  
  
        if (data.classIndex() == -1)  
            data.setClassIndex(data.numAttributes() - 1);  
  
        System.out.println(data);  
    }  
}
```

Screenshot of output:



```
Output - My_First_Ml_Model (run) ✘  
run:  
@relation pima_diabetes  
  
@attribute preg numeric  
@attribute plas numeric  
@attribute pres numeric  
@attribute skin numeric  
@attribute insu numeric  
@attribute mass numeric  
@attribute pedi numeric  
@attribute age numeric  
@attribute class {tested_negative,tested_positive}  
  
@data  
6,148,72,35,0,33.6,0.627,50,tested_positive  
1,85,66,29,0,26.6,0.351,31,tested_negative  
8,183,64,0,0,23.3,0.672,32,tested_positive  
1,89,66,23,94,28.1,0.167,21,tested_negative
```

Explanation:

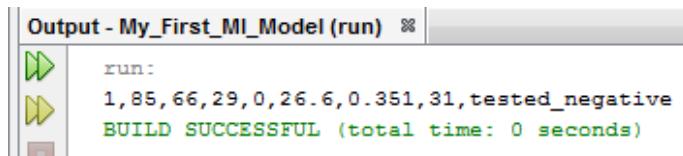
Definition:	System.out.println(data) is standard output stream that displays value of the data for output data is the object of the class Instances and it invokes the function getDataSet() to load the dataset into memory with the reference of data.
Code:	System.out.println(data);
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/core/Instances.html https://docs.oracle.com/javase/8/docs/api/java/lang/System.html

b. Display the values of the first instance from the dataset

Screenshot of code:

```
public class My_First_Ml_Model {  
    public static void main(String[] args) throws Exception {  
  
        DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");  
        Instances data = source.getDataSet();  
  
        if (data.classIndex() == -1)  
            data.setClassIndex(data.numAttributes() - 1);  
  
        System.out.println(data.instance(1));  
    }  
}
```

Screenshot of output:



The screenshot shows the Weka interface with the 'Output' tab selected. The output window displays the command 'run:' followed by the dataset instance: '1,85,66,29,0,26.6,0.351,31,tested_negative'. Below this, a green message indicates 'BUILD SUCCESSFUL (total time: 0 seconds)'.

Explanation:

Definition:	System.out.println(data) is standard output stream that displays value of the data for output data is the object of the class Instances and it invokes the function instance(1) by passing the parameter value '1' ('1' denotes the index number of instance of the dataset) to return the instance of index number '1' from the dataset.
Code:	System.out.println(data.instance(1));
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/core/Instances.html https://docs.oracle.com/javase/8/docs/api/java/lang/System.html

c. Display the first five instances from the dataset

Screenshot of code:

```
public class My_First_Ml_Model {  
    public static void main(String[] args) throws Exception {  
  
        DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");  
        Instances data = source.getDataSet();  
  
        if (data.classIndex() == -1)  
            data.setClassIndex(data.numAttributes() - 1);  
  
        for (int i=0; i<=4; i++)  
        {  
            System.out.println(data.instance(i));  
        }  
    }  
}
```

Screenshot of output:

```

Output - My_First_ML_Model (run) ✘
run:
6,148,72,35,0,33.6,0.627,50,tested_positive
1,85,66,29,0,26.6,0.351,31,tested_negative
8,183,64,0,0,23.3,0.672,32,tested_positive
1,89,66,23,94,28.1,0.167,21,tested_negative
0,137,40,35,168,43.1,2.288,33,tested_positive
BUILD SUCCESSFUL (total time: 0 seconds)

```

Explanation:

Definition:	System.out.println(data) is standard output stream that displays value of the data for output data is the object of the class Instances and it invokes the function instance(1) by passing the parameter value '1' ('1' denotes the index number of instance of the dataset) to return the instance of index number '1' from the dataset. for is the looping statement. Here, it helps to invoke the instances by passing the parameter 'i' for representing the index of the instance to the method instance(i).
Code:	<pre> for (int i=0; i<=4; i++) { System.out.println(data.instance(i)); } </pre>
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/core/Instances.html https://docs.oracle.com/javase/8/docs/api/java/lang/System.html https://docs.oracle.com/javase/8/docs/technotes/guides/language/foreach.html

d. Display the class value of the first instance

Screenshot of code:

```

public class My_First_ML_Model {
    public static void main(String[] args) throws Exception {

        DataSource source = new DataSource("C:\\\\Program Files\\\\Weka-3-8\\\\data\\\\diabetes.arff");
        Instances data = source.getDataSet();

        if (data.classIndex() == -1)
            data.setClassIndex(data.numAttributes() - 1);

        System.out.println(data.instance(1).classValue());
    }
}

```

Screenshot of output:

```

Output - My_First_ML_Model (run) ✘
run:
0.0
BUILD SUCCESSFUL (total time:

```

Explanation:

Definition:	System.out.println(data) is standard output stream that displays value of the data for output data is the object of the class Instances and it invokes the function instance(1) by passing the parameter value '1' ('1' denotes the index number of instance of the dataset) to return the instance of index number '1' from the dataset. classValue() returns the class value of a particular index of an instance.
Code:	System.out.println(data.instance(1).classValue());
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/core/Instances.html https://docs.oracle.com/javase/8/docs/api/java/lang/System.html

e. Display the total number of attributes present in the dataset

Screenshot of code:

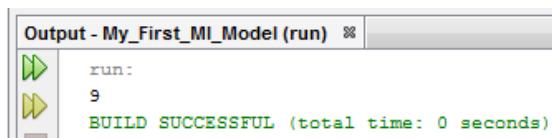
```
public class My_First_Ml_Model {
    public static void main(String[] args) throws Exception {

        DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
        Instances data = source.getDataSet();

        if (data.classIndex() == -1)
            data.setClassIndex(data.numAttributes() - 1);

        System.out.println(data.instance(1).numAttributes());
    }
}
```

Screenshot of output:



Explanation:

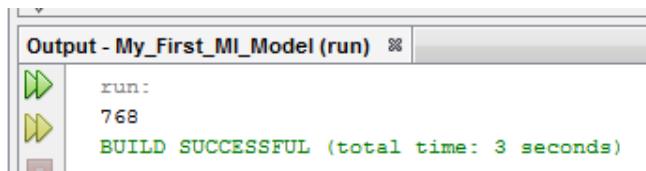
Definition:	data is the object of the class Instances and it invokes the function instance(1) by passing the parameter value '1' ('1' denotes the index number of instance of the dataset) to return the instance of index number '1' from the dataset. numAttributes() returns the total number of attributes present in the dataset.
Code:	System.out.println(data.numAttributes());
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/core/Instances.html https://docs.oracle.com/javase/8/docs/api/java/lang/System.html

f. Display the total number of instances present in the dataset

Screenshot of code:

```
public class My_First_Ml_Model {  
    public static void main(String[] args) throws Exception {  
  
        DataSource source = new DataSource("C:\\\\Program Files\\\\Weka-3-8\\\\data\\\\diabetes.arff");  
        Instances data = source.getDataSet();  
  
        if (data.classIndex() == -1)  
            data.setClassIndex(data.numAttributes() - 1);  
  
        System.out.println(data.numInstances());  
  
    }  
}
```

Screenshot of output:



Explanation:

Definition:	data is the object of the class Instances and it invokes the function instance(1) by passing the parameter value '1' ('1' denotes the index number of instance of the dataset) to return the instance of index number '1' from the dataset. numInstances() returns the total number of instances present in the dataset.
Code:	System.out.println(data.numInstances());
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/core/Instances.html https://docs.oracle.com/javase/8/docs/api/java/lang/System.html

2. Develop and display different types of Ml modes

a. Develop and display the tree-based J48 Ml model

Screenshot of code:

```
public class My_First_Ml_Model {  
    public static void main(String[] args) throws Exception {  
  
        DataSource source = new DataSource("C:\\\\Program Files\\\\Weka-3-8\\\\data\\\\diabetes.arff");  
        Instances data = source.getDataSet();  
  
        if (data.classIndex() == -1)  
            data.setClassIndex(data.numAttributes() - 1);  
  
        String[] options = new String[1];  
        options[0] = "-U";  
        J48 tree = new J48();  
        tree.setOptions(options);  
        tree.buildClassifier(data);  
  
        System.out.println(tree);  
  
    }  
}
```

Screenshot of output:

```
run:  
J48 unpruned tree  
-----  
  
plas <= 127  
|   mass <= 26.4  
|   |   preg <= 7: tested_negative (117.0/1.0)  
|   |   preg > 7  
|   |   |   mass <= 0: tested_positive (2.0)  
|   |   |   mass > 0: tested_negative (13.0)  
|   mass > 26.4  
|   |   age <= 28: tested_negative (180.0/22.0)  
|   |   age > 28  
|   |   |   plas <= 99: tested_negative (55.0/10.0)  
|   |   |   plas > 99  
|   |   |   |   pedi <= 0.56: tested_negative (84.0/34.0)  
|   |   |   |   pedi > 0.56  
|   |   |   |   preg <= 6
```

Explanation:

Definition:	tree is the object of the J48 class that invokes the buildClassifier(data) method to develop the J48 ML model
Code:	System.out.println(tree);
Ref. URL	http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html

b. Develop and display the naïve Bayes ML model

Screenshot of code:

```
import java.io.File;  
import weka.classifiers.trees.J48;  
import weka.core.converters.DataSource;  
import weka.classifiers.Evaluation;  
import java.util.Random;  
import weka.classifiers.bayes.NaiveBayes;  
import weka.classifiers.misc.SerializedClassifier;  
import weka.core.Debug;  
import weka.core.Instance;  
import weka.core.Instances;  
  
public class My_First_Ml_Model {  
    public static void main(String[] args) throws Exception {  
  
        DataSource source = new DataSource("C:\\\\Program Files\\\\Weka-3-8\\\\data\\\\diabetes.arff");  
        Instances data = source.getDataSet();  
  
        if (data.classIndex() == -1)  
            data.setClassIndex(data.numAttributes() - 1);  
  
        NaiveBayes nb = new NaiveBayes();  
        nb.buildClassifier(data);  
        System.out.println(nb);  
  
    }  
}
```

Screenshot of output:

Attribute	Class	
	tested_negative	tested_positive
(0.65)	(0.35)	
preg		
mean	3.4234	4.9795
std. dev.	3.0166	3.6827
weight sum	500	268
precision	1.0625	1.0625
plas		
mean	109.9541	141.2581
std. dev.	26.1114	31.8728
weight sum	500	268
precision	1.4741	1.4741
pres		
mean	68.1397	70.718
std. dev.	17.9834	21.4094
weight sum	500	268
precision	2.6522	2.6522
skin		

Explanation:

Definition:	<code>nb</code> is the object of the <code>NaiveBayes</code> class that invokes the <code>buildClassifier(data)</code> method to develop the <code>NaiveBayes</code> Ml model Note: the following naïve Bayes classifier package must be imported before run the code. <code>import weka.classifiers.bayes.NaiveBayes;</code>
Code:	<code>import weka.classifiers.bayes.NaiveBayes; System.out.println(nb);</code>
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/bayes/NaiveBayes.html

3. Perform the evaluation with different mode of test options

- Develop naïve Bayes Ml model and display the accuracy (percentage of instances correctly predicted) in 10 fold cross validation

Screenshot of code:

```
public class My_First_Ml_Model {  
    public static void main(String[] args) throws Exception {  
  
        DataSource source = new DataSource("C:\\\\Program Files\\\\Weka-3-8\\\\data\\\\diabetes.arff");  
        Instances data = source.getDataSet();  
  
        if (data.classIndex() == -1)  
            data.setClassIndex(data.numAttributes() - 1);  
    }  
}
```

```

NaiveBayes nb = new NaiveBayes();
nb.buildClassifier(data);

Evaluation eval = new Evaluation(data);
eval.crossValidateModel(nb, data, 10, new Random(1));

System.out.println(eval.pctCorrect());

```

Screenshot of output:

The screenshot shows the Weka interface with the title bar 'Output - My_First_Ml_Model (run)'. Below it, there are three entries: 'run:', '76.30208333333333', and 'BUILD SUCCESSFUL (total time: 1 second)'. Each entry has a small colored icon next to it: green for 'run:', yellow for '76.30208333333333', and brown for 'BUILD SUCCESSFUL'.

Explanation:

Definition:	eval is the object of the Evaluation class that invokes the crossValidateModel() method and pctCorrect() method. crossValidateModel(Classifier classifier, Instances data, int numFolds, java.util.Random random) performs a cross-validation for a classifier on a set of instances. crossValidateModel buildClas() to return the percentage of instances correctly classified (that is, for which a correct prediction was made).
Code:	<pre> import weka.classifiers.bayes.NaiveBayes; Evaluation eval = new Evaluation(data); eval. (nb, data, 10, new Random(1)); System.out.println(eval.pctCorrect()); System.out.println(nb); </pre>
Ref. URL	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

- b. Develop naïve Bayes Ml model and display the accuracy (percentage of instances correctly predicted) and use the test set as training set.**

Screenshot of code:

```

public class My_First_Ml_Model {
    public static void main(String[] args) throws Exception {

        DataSource source = new DataSource("C:\\\\Program Files\\\\Weka-3-8\\\\data\\\\diabetes.arff");
        Instances data = source.getDataSet();

        if (data.classIndex() == -1)
            data.setClassIndex(data.numAttributes() - 1);

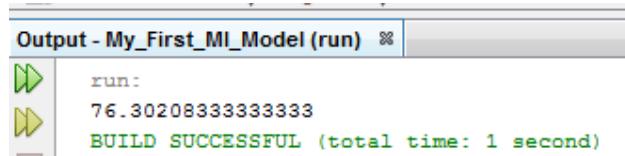
        NaiveBayes nb = new NaiveBayes();
        nb.buildClassifier(data);

        Evaluation eval = new Evaluation(data);
        eval.evaluateModel(nb, data);

        System.out.println(eval.pctCorrect());
    }
}

```

Screenshot of output:



```
Output - My_First_ML_Model (run) ✘
run:
76.30208333333333
BUILD SUCCESSFUL (total time: 1 second)
```

Explanation:

Definition:	eval.evaluateModel() method evaluates the classifier on a given set of instances.
Code:	eval.evaluateModel(nb, data);
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

4. Display the performance of the ML model using different performance metrics

- Develop naïve Bayes ML model and display the TP Rate for the class labels (test mode is use training set as test set)

Screenshot of code:

```
public class My_First_ML_Model {
    public static void main(String[] args) throws Exception {

        DataSource source = new DataSource("C:\\\\Program Files\\\\Weka-3-8\\\\data\\\\diabetes.arff");
        Instances data = source.getDataSet();

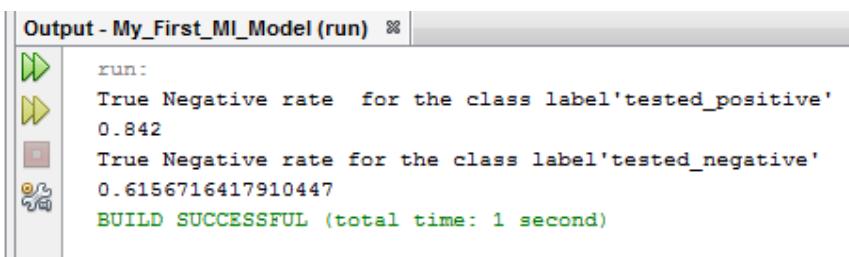
        if (data.classIndex() == -1)
            data.setClassIndex(data.numAttributes() - 1);

        NaiveBayes nb = new NaiveBayes();
        nb.buildClassifier(data);

        Evaluation eval = new Evaluation(data);
        eval.evaluateModel(nb, data);
        System.out.println("True Negative rate for the class label" + "'tested_positive'");
        System.out.println(eval.trueNegativeRate(1));
        System.out.println("True Negative rate for the class label" + "'tested_negative'");
        System.out.println(eval.trueNegativeRate(0));

    }
}
```

Screenshot of output:



```
Output - My_First_ML_Model (run) ✘
run:
True Negative rate for the class label'tested_positive'
0.842
True Negative rate for the class label'tested_negative'
0.6156716417910447
BUILD SUCCESSFUL (total time: 1 second)
```

Explanation:

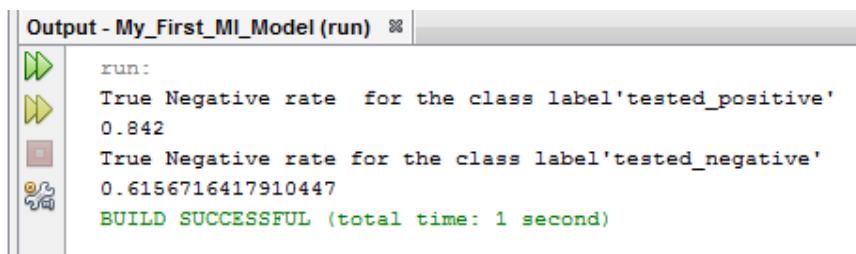
Definition:	truePositiveRate(int classIndex) calculates the true positive rate with respect to a particular class.
Code:	Eval.truePositiveRate (1) Eval.truePositiveRate (0)
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

- b. Develop naïve Bayes ML model and display the TN Rate for the class labels (test mode is use training set as test set)**

Screenshot of code:

```
public class My_First_Ml_Model {  
    public static void main(String[] args) throws Exception {  
  
        DataSource source = new DataSource("C:\\\\Program Files\\\\Weka-3-8\\\\data\\\\diabetes.arff");  
        Instances data = source.getDataSet();  
  
        if (data.classIndex() == -1)  
            data.setClassIndex(data.numAttributes() - 1);  
  
        NaiveBayes nb = new NaiveBayes();  
        nb.buildClassifier(data);  
  
        Evaluation eval = new Evaluation(data);  
        eval.evaluateModel(nb, data);  
        System.out.println("True Negative rate for the class label" + "'tested_positive'");  
        System.out.println(eval.trueNegativeRate(1));  
        System.out.println("True Negative rate for the class label" + "'tested_negative'");  
        System.out.println(eval.trueNegativeRate(0));  
    }  
}
```

Screenshot of output:



```
Output - My_First_Ml_Model (run) ***  
run:  
True Negative rate for the class label'tested_positive'  
0.842  
True Negative rate for the class label'tested_negative'  
0.6156716417910447  
BUILD SUCCESSFUL (total time: 1 second)
```

Explanation:

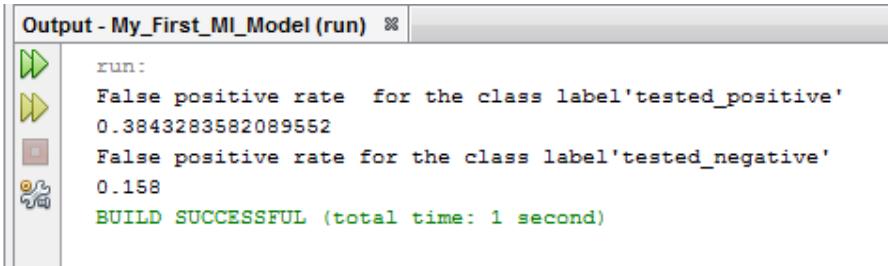
Definition:	trueNegativeRate(int classIndex) calculates the true negative rate with respect to a particular.
Code:	Eval.trueNegativeRate(1) eval.trueNegativeRate(0)
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

- c. Develop naïve Bayes ML model and display the Develop naïve Bayes ML model and display the FP for the class labels (test mode is use training set as test set)

Screenshot of code:

```
public class My_First_ML_Model {  
    public static void main(String[] args) throws Exception {  
  
        DataSource source = new DataSource("C:\\\\Program Files\\\\Weka-3-8\\\\data\\\\diabetes.arff");  
        Instances data = source.getDataSet();  
  
        if (data.classIndex() == -1)  
            data.setClassIndex(data.numAttributes() - 1);  
  
        NaiveBayes nb = new NaiveBayes();  
        nb.buildClassifier(data);  
  
        Evaluation eval = new Evaluation(data);  
        eval.evaluateModel(nb, data);  
        System.out.println("False positive rate for the class label" + "'tested_positive'");  
        System.out.println(eval.falseNegativeRate(1));  
        System.out.println("False positive rate for the class label" + "'tested_negative'");  
        System.out.println(eval.falseNegativeRate(0));  
    }  
}
```

Screenshot of output:



```
Output - My_First_ML_Model (run) ✘  
run:  
    False positive rate for the class label'tested_positive'  
    0.3843283582089552  
    False positive rate for the class label'tested_negative'  
    0.158  
    BUILD SUCCESSFUL (total time: 1 second)
```

Explanation:

Definition	falsePositiveRate(int classIndex) calculate the false positive rate with respect to a particular class.
Part of the Code:	System.out.println(eval.falseNegativeRate(1)); System.out.println(eval.falseNegativeRate(0));
Ref. URL	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

- d. Develop naïve Bayes ML model and display the Develop naïve Bayes ML model and display the FN Rate for the class labels (test mode is use training set as test set)

Screenshot of code:

```
public class My_First_ML_Model {  
    public static void main(String[] args) throws Exception {
```

```
DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();

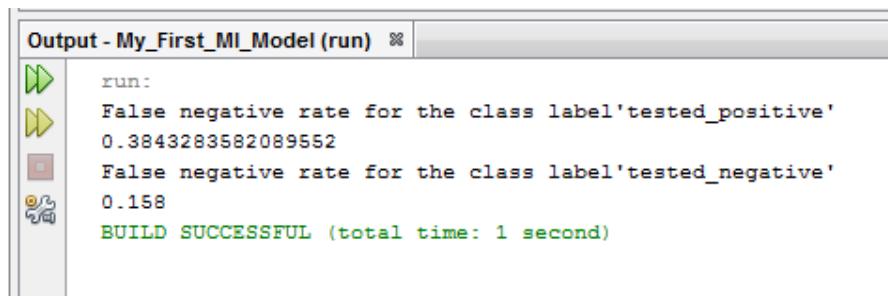
if (data.classIndex() == -1)
    data.setClassIndex(data.numAttributes() - 1);

NaiveBayes nb = new NaiveBayes();
nb.buildClassifier(data);

Evaluation eval = new Evaluation(data);
eval.evaluateModel(nb, data);
System.out.println("False negative rate for the class label" + "'tested_positive'");
System.out.println(eval.falseNegativeRate(1));
System.out.println("False negative rate for the class label" + "'tested_negative'");
System.out.println(eval.falseNegativeRate(0));

}}
```

Screenshot of output:



```
run:
False negative rate for the class label'tested_positive'
0.3843283582089552
False negative rate for the class label'tested_negative'
0.158
BUILD SUCCESSFUL (total time: 1 second)
```

Explanation:

Definition:	falseNegativeRate(int classIndex) method calculates the false negative rate with respect to a particular class.
Code:	eval.falseNegativeRate(1) eval.falseNegativeRate(0)
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

- a. Develop naïve Bayes Ml model and display the Precision for the class labels (test mode is use training set as test set)

Screenshot of code:

```
public class My_First_Ml_Model {
    public static void main(String[] args) throws Exception {

        DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
        Instances data = source.getDataSet();

        if (data.classIndex() == -1)
            data.setClassIndex(data.numAttributes() - 1);

        NaiveBayes nb = new NaiveBayes();
        nb.buildClassifier(data);
```

```
Evaluation eval = new Evaluation(data);
eval.evaluateModel(nb, data);
System.out.println("Precision for the class label" + "'tested_positive'");
System.out.println(eval.precision(1));
System.out.println("Precision for the class label" + "'tested_negative'");
System.out.println(eval.precision(0));
}

}}
```

Screenshot of output:

```
Output - My_First_Ml_Model (run) ✘
run:
Precision for the class label'tested_positive'
0.6762295081967213
Precision for the class label'tested_negative'
0.8034351145038168
BUILD SUCCESSFUL (total time: 1 second)
```

Explanation:

Definition:	precision(int classIndex) method calculates the precision with respect to a particular class.
Code:	System.out.println(eval.precision(1)); System.out.println(eval.precision(0));
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

- a. Develop naïve Bayes MI model and display the accuracy (percentage of the correctly predicted instance) for the class labels (test mode is use training set as test set)

Screenshot of code:

```
public class My_First_Ml_Model {
    public static void main(String[] args) throws Exception {

        DataSource source = new DataSource("C:\\\\Program Files\\\\Weka-3-8\\\\data\\\\diabetes.arff");
        Instances data = source.getDataSet();

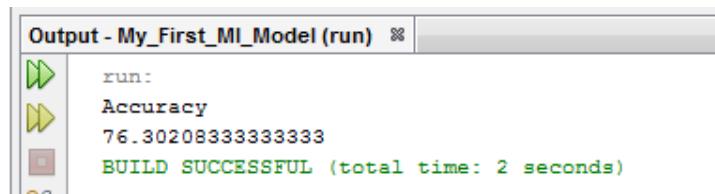
        if (data.classIndex() == -1)
            data.setClassIndex(data.numAttributes() - 1);

        NaiveBayes nb = new NaiveBayes();
        nb.buildClassifier(data);

        Evaluation eval = new Evaluation(data);
        eval.evaluateModel(nb, data);
        System.out.println("Accuracy ");

        System.out.println(eval.pctCorrect());
    }
}
```

Screenshot of output:



```
Output - My_First_ML_Model (run) ✘
run:
Accuracy
76.30208333333333
BUILD SUCCESSFUL (total time: 2 seconds)
```

Explanation:

Definition:	pctCorrect() returns the percentage of instances correctly classified (that is, for which a correct prediction was made).
Code:	System.out.println(eval.pctCorrect());
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

b. Develop naïve Bayes Ml model and display the Precision for the class labels (test mode is use training set as test set)

Screenshot of code:

```
public class My_First_ML_Model {
    public static void main(String[] args) throws Exception {

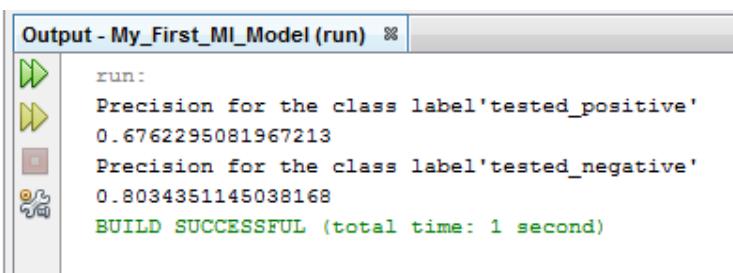
        DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
        Instances data = source.getDataSet();

        if (data.classIndex() == -1)
            data.setClassIndex(data.numAttributes() - 1);

        NaiveBayes nb = new NaiveBayes();
        nb.buildClassifier(data);

        Evaluation eval = new Evaluation(data);
        eval.evaluateModel(nb, data);
        System.out.println("Precision for the class label" + "'tested_positive'");
        System.out.println(eval.precision(1));
        System.out.println("Precision for the class label" + "'tested_negative'");
        System.out.println(eval.precision(0));
    }
}
```

Screenshot of output:



```
Output - My_First_ML_Model (run) ✘
run:
Precision for the class label'tested_positive'
0.6762295081967213
Precision for the class label'tested_negative'
0.8034351145038168
BUILD SUCCESSFUL (total time: 1 second)
```

Explanation:

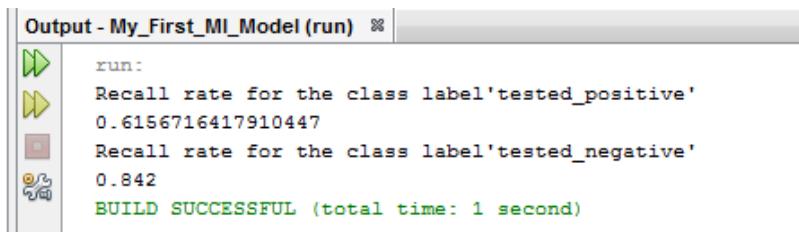
Definition:	precision(int classIndex) method calculates the precision with respect to a particular class.
Code:	System.out.println(eval.precision(1));
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

- e. Develop naïve Bayes ML model and display the Recall for the class labels (test mode is use training set as test set)

Screenshot of code:

```
public class My_First_ML_Model {  
    public static void main(String[] args) throws Exception {  
  
        DataSource source = new DataSource("C:\\\\Program Files\\\\Weka-3-8\\\\data\\\\diabetes.arff");  
        Instances data = source.getDataSet();  
  
        if (data.classIndex() == -1)  
            data.setClassIndex(data.numAttributes() - 1);  
  
        NaiveBayes nb = new NaiveBayes();  
        nb.buildClassifier(data);  
  
        Evaluation eval = new Evaluation(data);  
        eval.evaluateModel(nb, data);  
        System.out.println("Recall rate for the class label" + "'tested_positive'");  
  
        System.out.println(eval.recall(1));  
        System.out.println("Recall rate for the class label" + "'tested_negative'");  
        System.out.println(eval.recall(0));  
    }  
}
```

Screenshot of output:



```
Output - My_First_ML_Model (run) *  
run:  
Recall rate for the class label'tested_positive'  
0.6156716417910447  
Recall rate for the class label'tested_negative'  
0.842  
BUILD SUCCESSFUL (total time: 1 second)
```

Explanation:

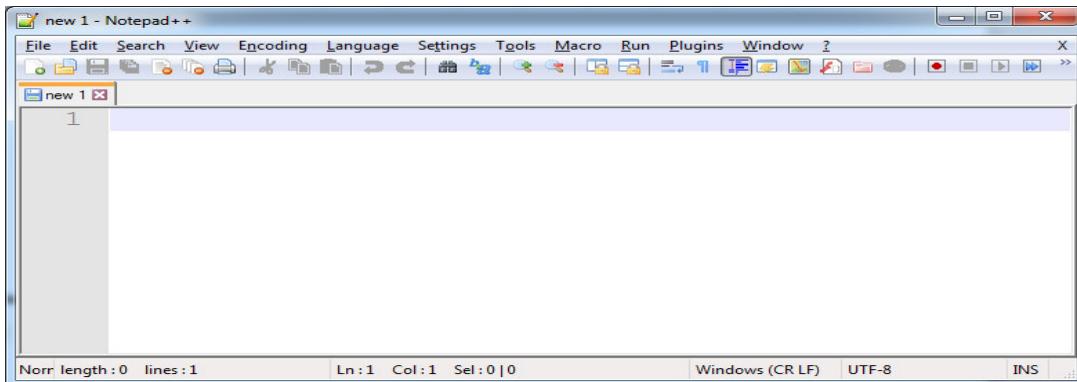
Definition:	recall(int classIndex) method Calculate the recall with respect to a particular class.
Code:	System.out.println(eval.recall(1));
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

Chapter 4 **Prediction using ML model**

Part I

Preparing the unknown (unlabeled) dataset

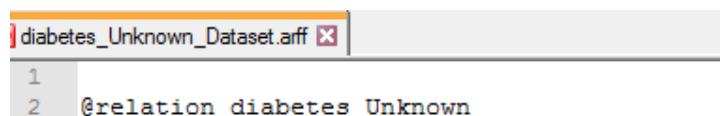
1. Open Notepad/Notepad++ and



2. Define the name of the dataset:

Syntax: @ relation <dataset_name>

Code: @relation diabetes_Unknown



3. Define the attributes and class label of the dataset:

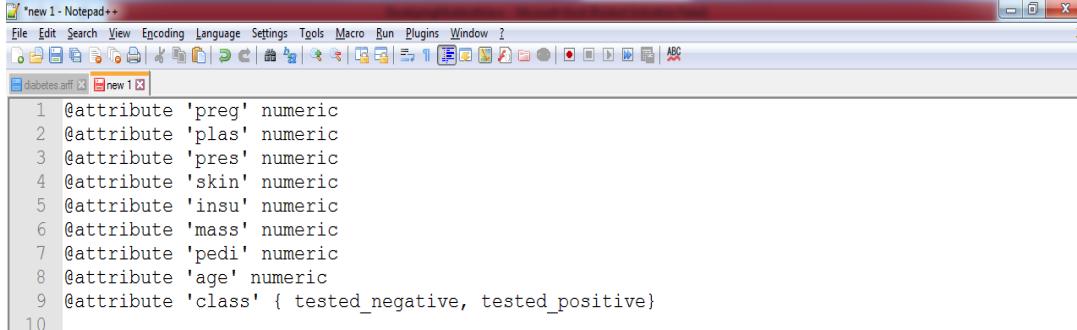
Syntax: @ attribute <name of the attribute > <attribute type>

Syntax: @ attribute <class label name > <attribute type>

Code: @attribute 'preg' numeric

```
@attribute 'plas' numeric  
@attribute 'pres' numeric  
@attribute 'skin' numeric  
@attribute 'insu' numeric  
@attribute 'mass' numeric  
@attribute 'pedi' numeric  
@attribute 'age' numeric
```

```
@attribute 'class' { tested_negative, tested_positive}
```



4. Define the instances of the dataset:

Syntax:

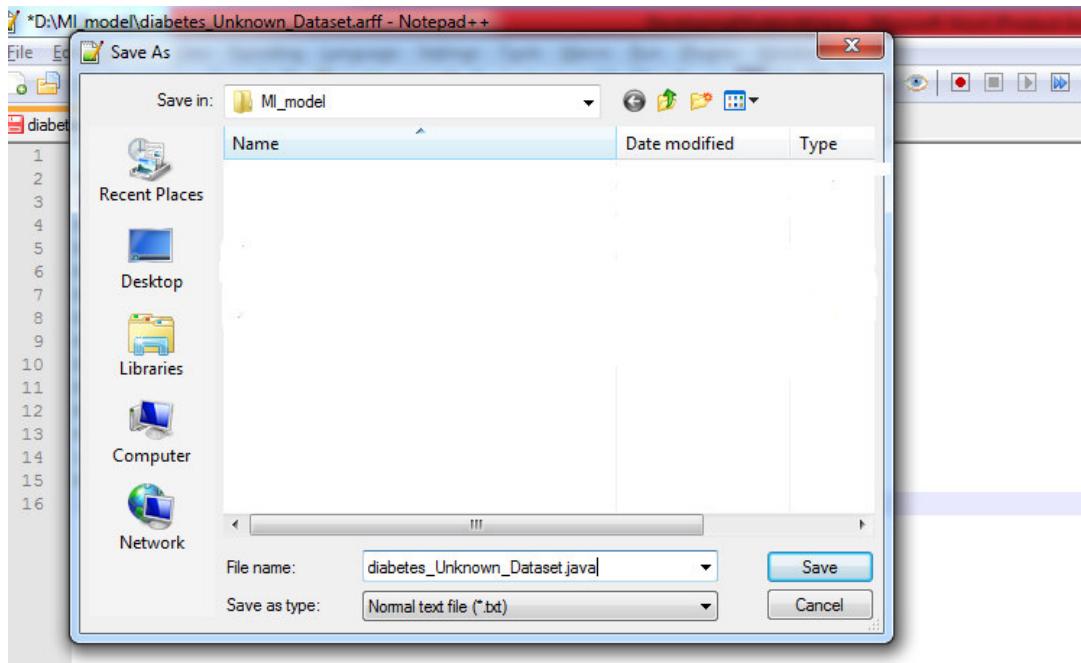
```
@data  
< value for attribute 1, value for attribute 2, .... value for attribute n>
```

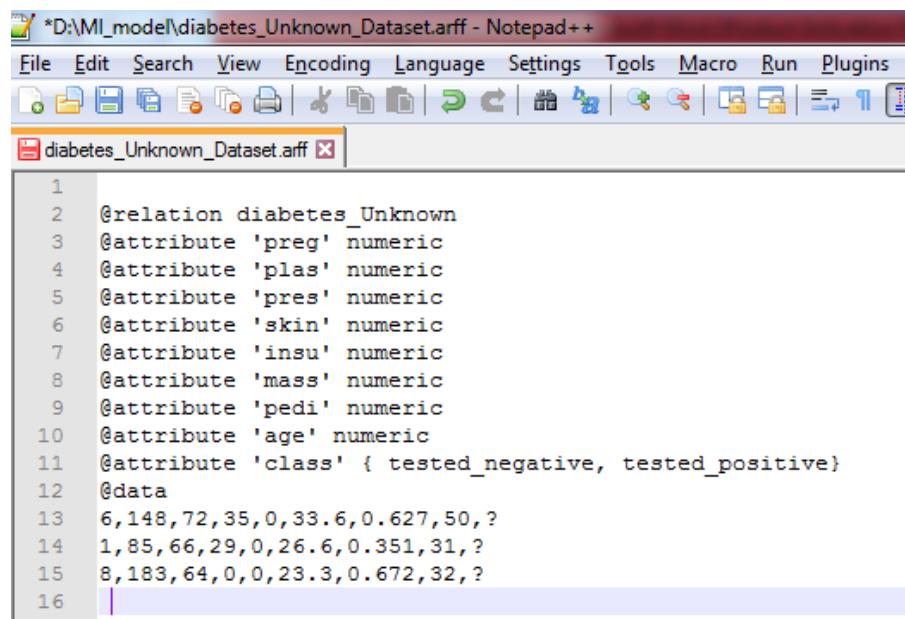
Code:

```
@data  
6,148,72,35,0,33.6,0.627,50,?  
1,85,66,29,0,26.6,0.351,31,?  
8,183,64,0,0,23.3,0.672,32,?
```

```
@relation diabetes_Unknown  
@attribute 'preg' numeric  
@attribute 'plas' numeric  
@attribute 'pres' numeric  
@attribute 'skin' numeric  
@attribute 'insu' numeric  
@attribute 'mass' numeric  
@attribute 'pedi' numeric  
@attribute 'age' numeric  
@attribute 'class' { tested_negative, tested_positive}  
@data  
6,148,72,35,0,33.6,0.627,50,?  
1,85,66,29,0,26.6,0.351,31,?  
8,183,64,0,0,23.3,0.672,32,?
```

5. Save the file with the file extension of arff shown in screenshots below:





The screenshot shows the Notepad++ interface with the file 'diabetes_Unknown_Dataset.arff' open. The window title is 'D:\MI_model\diabetes_Unknown_Dataset.arff - Notepad++'. The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Tools, Macro, Run, Plugins. The toolbar has various icons for file operations like Open, Save, Print, Find, Replace, etc. The code editor displays the following ARFF file content:

```
1
2 @relation diabetes_Unknown
3 @attribute 'preg' numeric
4 @attribute 'plas' numeric
5 @attribute 'pres' numeric
6 @attribute 'skin' numeric
7 @attribute 'insu' numeric
8 @attribute 'mass' numeric
9 @attribute 'pedi' numeric
10 @attribute 'age' numeric
11 @attribute 'class' { tested_negative, tested_positive}
12 @data
13 6,148,72,35,0,33.6,0.627,50,?
14 1,85,66,29,0,26.6,0.351,31,?
15 8,183,64,0,0,23.3,0.672,32,?
16 |
```

Part II Developing the MI Models and Perform Prediction

Outline of the program

The steps to developed the java program is shown in the below Figure. Initially, the required packages are imported. Machine learning model is developed from the loaded training dataset and the model is saved in the local disk or cloud. Then, model and unknown (unlabeled) dataset are loaded. The prediction is performed using the model on the unknown dataset and the predicted results are displayed.

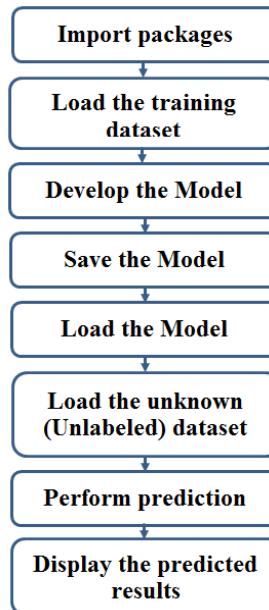


Figure: Steps to develop a Java code for performing prediction using MI models

Step I: Develop the Java code to construct and predict label of the unknown data MI models

1. Import the required packages:

a. Import package to handle the files

Definition:	File(String pathname) creates a new File instance by converting the given pathname string into an abstract pathname.
Code:	import java.io.File;
Ref. URL	https://docs.oracle.com/javase/8/docs/api/java/io/File.html

b. Import Package to develop the MI model (J48)

Definition:	J48 class is used to generate decision tree-based machine learning model.
Code:	import weka.classifiers.trees.J48;
Ref. URL	http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html

c. Import package to access the dataset from a file

Definition:	DataSource class is used loading data from files
Code:	Import weka.core.converters.ConverterUtils.DataSource;
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/core/converters/ConverterUtils.DataSource.html

d. Import package to load the MI models

Definition:	SerializedClassifier loads serialized models and uses it to make predictions.
Code:	import weka.classifiers.misc.SerializedClassifier;
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/classifiers/misc/SerializedClassifier.html

e. Import package for debugging

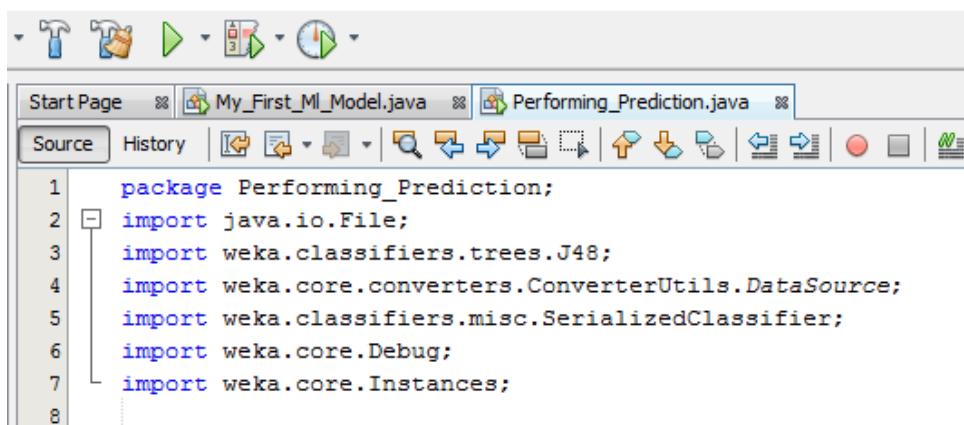
Definition:	Debug class is used for debug output, logging, clocking, etc.
Code:	import weka.core.Debug;
Ref. URL	http://weka.sourceforge.net/doc.dev/weka/core/Debug.html

f. Import package to handle the instance of the dataset

Definition	Instances class is used to handle the ordered set of weighted instances.
Code:	import weka.core Instances;
Ref. URL	http://weka.sourceforge.net/doc.dev/weka/core/Instances.html

Snapshot of code to import required classes:

```
import java.io.File;
import weka.classifiers.trees.J48;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.misc.SerializedClassifier;
import weka.core.Debug;
import weka.core.Instances;
```



Step II: Define the Main class and the Main method

Note: Below a Java code is provided for displaying the string "HelloWorld"

```
public class HelloWorldApp {  
    public static void main(String[] args) {  
        System.out.println("Hello World!"); // Display the  
        string.  
    }  
}
```

- public class: Any other class can access a public field or method.
- public method: This method is public and therefore available to anyone.
- static method: This method can be run without having to create an instance of the class MyClass.
- void method: This method does not return anything.
- (String[] args): This method takes a String argument. Note that the argument args can be anything — it's common to use "args" but we could instead call it "stringArray".

Ref. URL: <https://docs.oracle.com/javase/tutorial/getStarted/application/index.html>

Definition:	public class: Any other class can access a public field or method. public method: This method is public and therefore available to anyone. static method: This method can be run without having to create an instance of the class MyClass. void method: This method does not return anything. (String[] args): This method takes a String argument. Note that the argument args can be anything — it's common to use "args" but we could instead call it "stringArray". The throws keyword indicates what exception type may be thrown by a method.
Code:	public class Performing_Prediction{ public static void main(String[] args) throws Exception {
Ref. URL:	https://docs.oracle.com/javase/tutorial/getStarted/application/index.html

Snapshot of code to define the Main class and the Main method

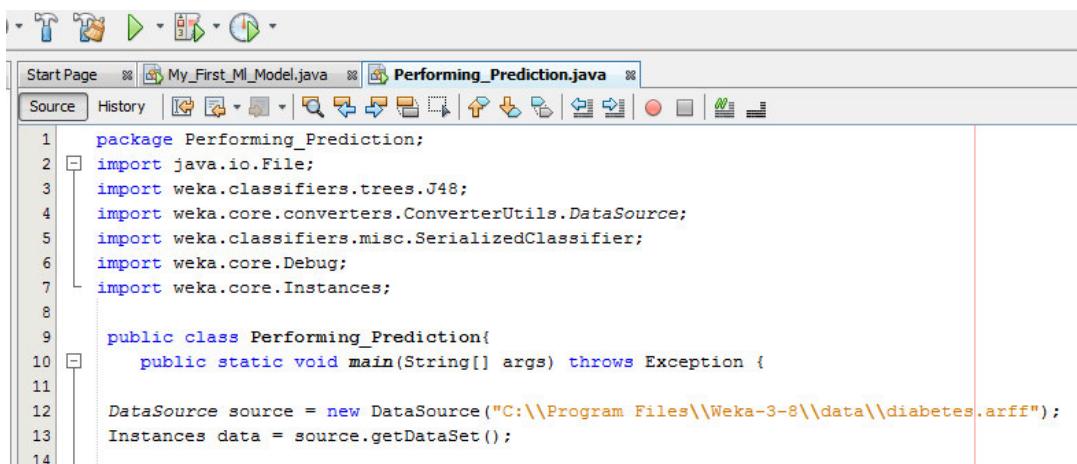
```
public class Performing_Prediction{  
    public static void main(String[] args) throws Exception {  
  
        package Performing_Prediction;  
        import java.io.File;  
        import weka.classifiers.trees.J48;  
        import weka.core.converters.ConverterUtils.DataSource;  
        import weka.classifiers.misc.SerializedClassifier;  
        import weka.core.Debug;  
        import weka.core.Instances;  
  
        public class Performing_Prediction{  
            public static void main(String[] args) throws Exception {
```

Step III: Load the dataset from the file

Definition:	DataSource class is used to loading data from files getDataSet() method returns the full dataset, can be null in case of an error. In this code, an object source is created for the class DataSource and the object source is used to invoke the method getDataSet() using dot operator Note: Provide the entire path of the dataset which should be loaded
Code:	DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff"); Instances data = source.getDataSet();
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/core/converter/ConverterUtils.DataSource.html

Snapshot of code to load the full dataset

```
DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");  
Instances data = source.getDataSet();
```

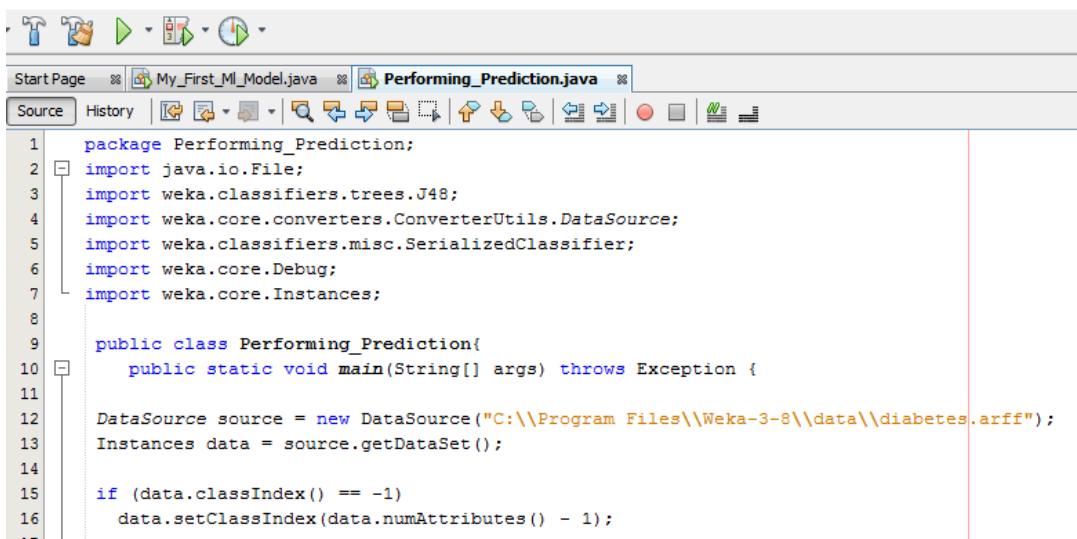


Step IV: Set the class attribute for the dataset

Definition:	classIndex() Returns the class attribute's index (default it is set -1). setClassIndex(int classIndex) Sets the class index of the dataset. numAttributes() Returns the total number of attributes of the dataset Note: all the above methods are invoked by the object data which is created for the class Instances.
Code:	if (data.classIndex() == -1) data.setClassIndex(data.numAttributes() - 1);
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/core/converter/ConverterUtils.DataSource.html

Snapshot of code to set the class attribute for the dataset

```
if (data.classIndex() == -1)  
    data.setClassIndex(data.numAttributes() - 1);
```



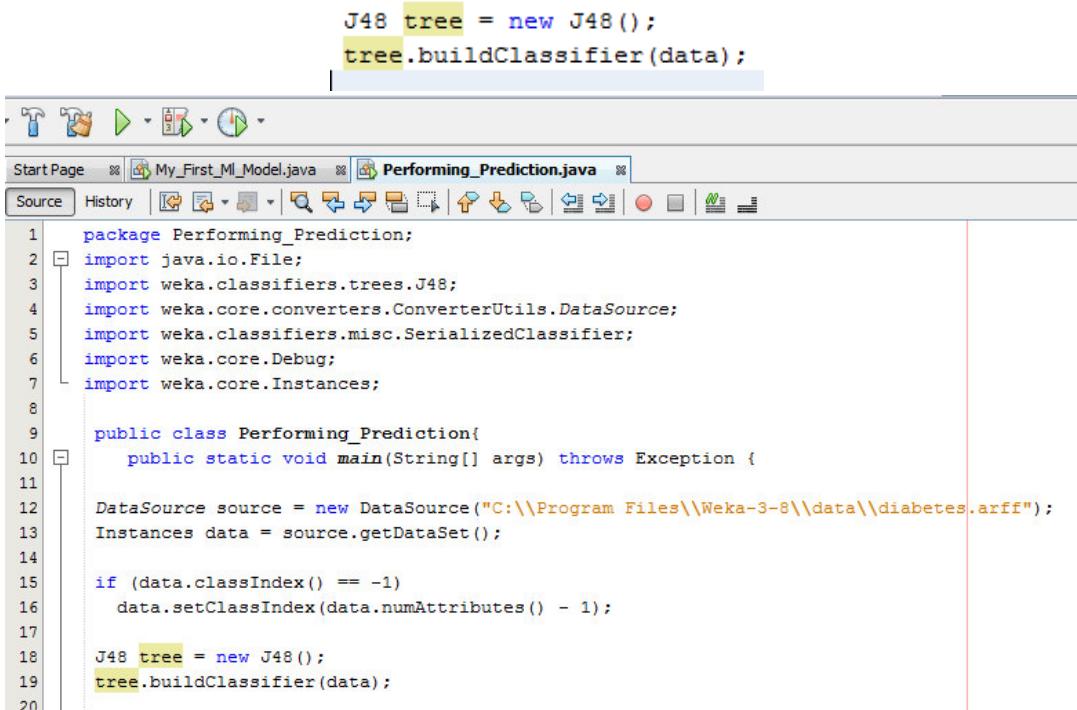
```

1 package Performing_Prediction;
2 import java.io.File;
3 import weka.classifiers.trees.J48;
4 import weka.core.converters.ConverterUtils.DataSource;
5 import weka.classifiers.misc.SerializedClassifier;
6 import weka.core.Debug;
7 import weka.core.Instances;
8
9 public class Performing_Prediction{
10     public static void main(String[] args) throws Exception {
11
12     DataSource source = new DataSource("C:\\\\Program Files\\\\Weka-3-8\\\\data\\\\diabetes.arff");
13     Instances data = source.getDataSet();
14
15     if (data.classIndex() == -1)
16         data.setClassIndex(data.numAttributes() - 1);
17
18 }
19 }
```

Step V: Develop the Machine Learning Model from the dataset

Definition:	J48 class for developing a pruned or unpruned C4.5 decision tree buildClassifier method generates the classifier (Machine learning model) Note: tree is the object of the class J48 and the data is the object of the class DataSource
Code:	J48 nb = new J48(); nb.buildClassifier(data);
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html

Snapshot of code to develop the Machine Learning Model from the dataset



```

J48 tree = new J48();
tree.buildClassifier(data);

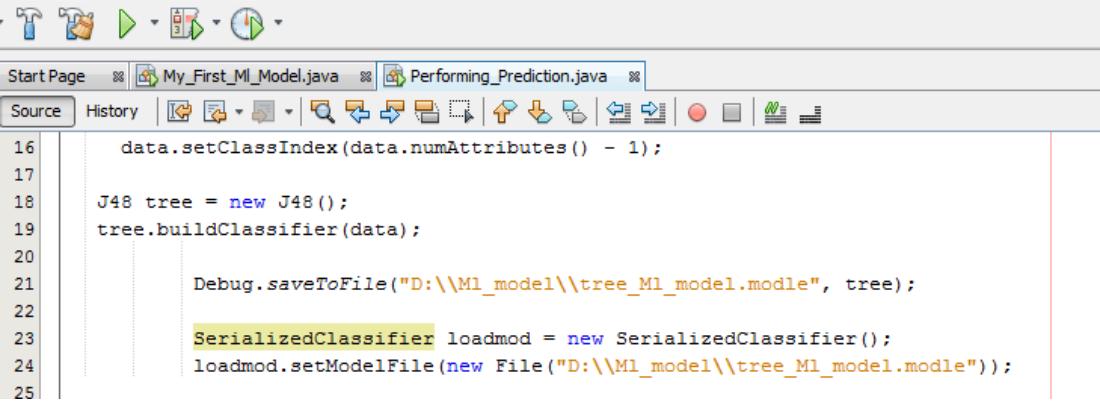
1 package Performing_Prediction;
2 import java.io.File;
3 import weka.classifiers.trees.J48;
4 import weka.core.converters.ConverterUtils.DataSource;
5 import weka.classifiers.misc.SerializedClassifier;
6 import weka.core.Debug;
7 import weka.core.Instances;
8
9 public class Performing_Prediction{
10     public static void main(String[] args) throws Exception {
11
12     DataSource source = new DataSource("C:\\\\Program Files\\\\Weka-3-8\\\\data\\\\diabetes.arff");
13     Instances data = source.getDataSet();
14
15     if (data.classIndex() == -1)
16         data.setClassIndex(data.numAttributes() - 1);
17
18     J48 tree = new J48();
19     tree.buildClassifier(data);
20 }
```

Step VI: Save the model

Definition:	Debug class is used for debug output, logging, clocking, etc. saveToFile () writes the serialized object to the specified file
Code:	Debug.saveToFile ("D:\\Ml_model\\nb_Ml_model.modle", tree);
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/core/Debug.html

Snapshot of code to evaluate the performance of Machine learning model and display the result in console window

```
Debug.saveToFile("D:\\Ml_model\\tree_Ml_model.modle", tree);
```



```

16     data.setClassIndex(data.numAttributes() - 1);
17
18     J48 tree = new J48();
19     tree.buildClassifier(data);
20
21     Debug.saveToFile("D:\\Ml_model\\tree_Ml_model.modle", tree);
22
23     SerializedClassifier loadmod = new SerializedClassifier();
24     loadmod.setModelFile(new File("D:\\Ml_model\\tree_Ml_model.modle"));
25

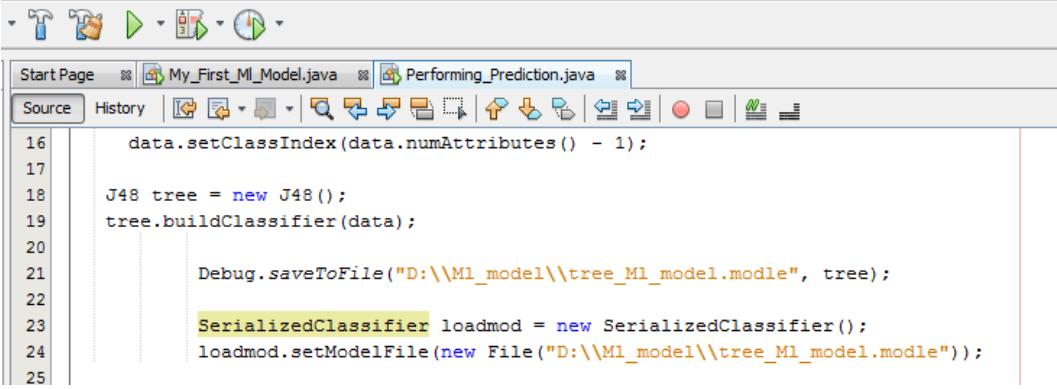
```

Step III: Load the model

Definition:	SerializedClassifier loads serialized models and uses it to make predictions. setModelFile(java.io.File value) sets the file containing the serialized model. File(String pathname) creates a new File instance by converting the given pathname string into an abstract pathname.
Code:	SerializedClassifier loadmod = new SerializedClassifier(); loadmod.setModelFile(new File("D:\\Ml_model\\tree_Ml_model.modle"));
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/misc/SerializedClassifier.html https://docs.oracle.com/javase/8/docs/api/java/io/File.html

Snapshot of code to load the full dataset

```
SerializedClassifier loadmod = new SerializedClassifier();
loadmod.setModelFile(new File("D:\\Ml_model\\tree_Ml_model.modle"));
```



```

16     data.setClassIndex(data.numAttributes() - 1);
17
18     J48 tree = new J48();
19     tree.buildClassifier(data);
20
21     Debug.saveToFile("D:\\Ml_model\\tree_Ml_model.modle", tree);
22
23     SerializedClassifier loadmod = new SerializedClassifier();
24     loadmod.setModelFile(new File("D:\\Ml_model\\tree_Ml_model.modle"));
25

```

Step III: Load the unknown dataset from the file

Definition:	DataSource class is used to loading data from files getDataSet() method returns the full dataset, can be null in case of an error. In this code, an object source1 is created for the class DataSource and the object source1 is used to invoke the method getDataSet() using dot operator Note: Provide the entire path of the dataset which should be loaded
Code:	DataSource source1 = new DataSource("D:\\Ml_model\\diabetes_Unknown_Dataset.arff"); Instances data1 = source1.getDataSet(); if (data1.classIndex() == -1 data1.setClassIndex(data1.numAttributes() - 1);
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/core/converter/ConverterUtils.DataSource.html

Snapshot of code to load the full dataset

```
DataSource source1 = new DataSource("D:\\Ml_model\\diabetes_Unknown_Dataset.arff");
Instances data1 = source1.getDataSet();

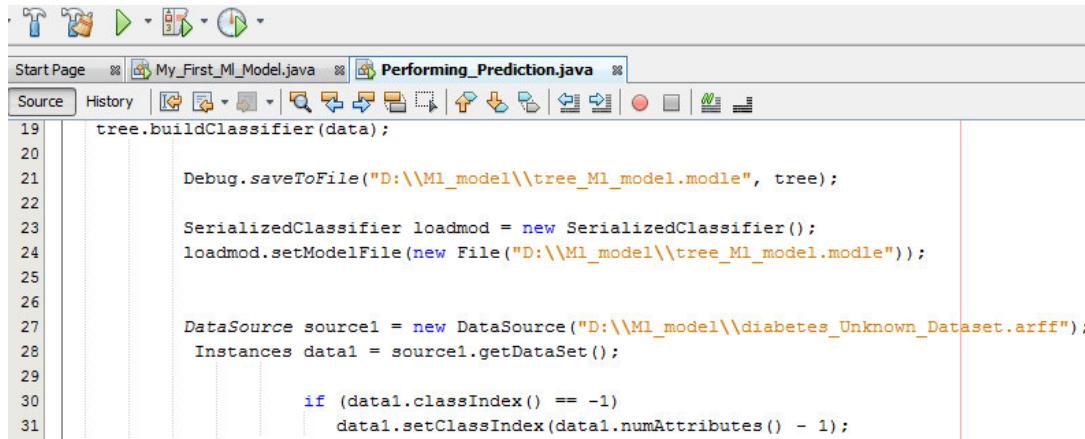
19   tree.buildClassifier(data);
20
21   Debug.saveToFile("D:\\Ml_model\\tree_Ml_model.modle", tree);
22
23   SerializedClassifier loadmod = new SerializedClassifier();
24   loadmod.setModelFile(new File("D:\\Ml_model\\tree_Ml_model.modle"));
25
26
27   DataSource source1 = new DataSource("D:\\Ml_model\\diabetes_Unknown_Dataset.arff");
28   Instances data1 = source1.getDataSet();
```

Step IV: Set the class attribute for the dataset

Definition:	classIndex() Returns the class attribute's index (default it is set -1). setClassIndex(int classIndex) Sets the class index of the dataset. numAttributes() Returns the total number of attributes of the dataset Note: all the above methods are invoked by the object data1 which is created for the class Instances.
Code:	if (data1.classIndex() == -1) data1.setClassIndex(data1.numAttributes() - 1);
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/core/converter/ConverterUtils.DataSource.html

Snapshot of code to load the full dataset

```
if (data1.classIndex() == -1)
    data1.setClassIndex(data1.numAttributes() - 1);
```



```

19     tree.buildClassifier(data);
20
21     Debug.saveToFile("D:\\Ml_model\\tree_Ml_model.modle", tree);
22
23     SerializedClassifier loadmod = new SerializedClassifier();
24     loadmod.setModelFile(new File("D:\\Ml_model\\tree_Ml_model.modle"));
25
26
27     DataSource source1 = new DataSource("D:\\Ml_model\\diabetes_Unknown_Dataset.arff");
28     Instances data1 = source1.getDataSet();
29
30     if (data1.classIndex() == -1)
31         data1.setClassIndex(data1.numAttributes() - 1);

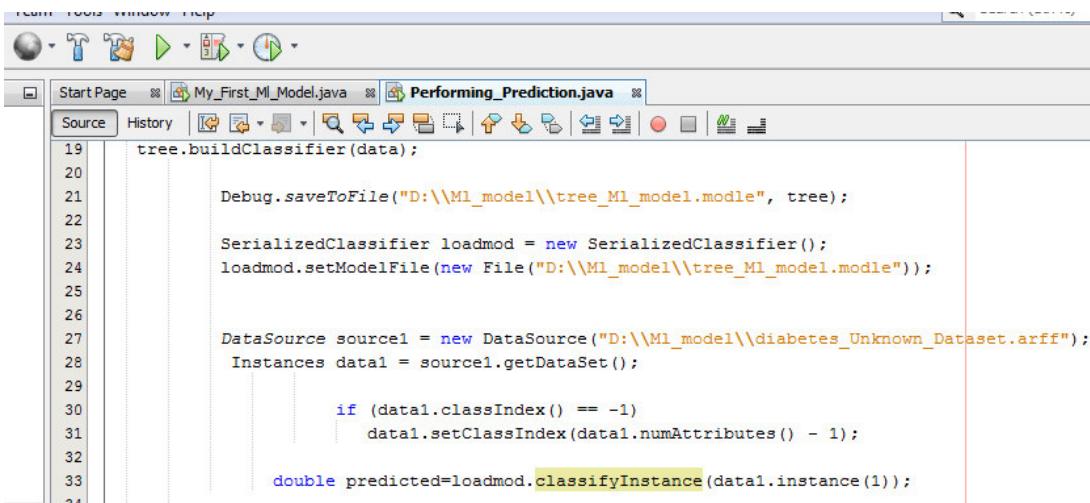
```

Step IV: Perform the prediction

Definition:	classifyInstance(Instance instance) classifies the given test instance. instance(int index) returns the instance at the given position.
Code:	double predicted=loadmod.classifyInstance(data1.instance(1));
Ref. URL	http://weka.sourceforge.net/doc.dev/weka/classifiers/Classifier.html http://weka.sourceforge.net/doc.dev/weka/core/Instances.html

Snapshot of code to set the class attribute for the dataset

```
double predicted=loadmod.classifyInstance(data1.instance(1));
```



```

19     tree.buildClassifier(data);
20
21     Debug.saveToFile("D:\\Ml_model\\tree_Ml_model.modle", tree);
22
23     SerializedClassifier loadmod = new SerializedClassifier();
24     loadmod.setModelFile(new File("D:\\Ml_model\\tree_Ml_model.modle"));
25
26
27     DataSource source1 = new DataSource("D:\\Ml_model\\diabetes_Unknown_Dataset.arff");
28     Instances data1 = source1.getDataSet();
29
30     if (data1.classIndex() == -1)
31         data1.setClassIndex(data1.numAttributes() - 1);
32
33     double predicted=loadmod.classifyInstance(data1.instance(1));
34

```

Step VI: Display the predicted label

Definition	System.out.println(predicted) is standard output stream that displays value of the variable predicted for output
Code:	System.out.println("Predicted label"); System.out.println(predicted);
Ref. URL	http://weka.sourceforge.net/doc.dev/weka/core/Debug.html

Snapshot of code to evaluate the performance of Machine learning model and display the result in console window

The screenshot shows a Java IDE interface with two tabs open: "My_First_Ml_Model.java" and "Performing_Prediction.java". The "Performing_Prediction.java" tab is active, displaying the following code:

```
System.out.println("Predicted label");
System.out.println(predicted);

tree.buildClassifier(data);

Debug.saveToFile("D:\\Ml_model\\tree_Ml_model.modle", tree);

SerializedClassifier loadmod = new SerializedClassifier();
loadmod.setModelFile(new File("D:\\Ml_model\\tree_Ml_model.modle"));

DataSource source1 = new DataSource("D:\\Ml_model\\diabetes_Unknown_Dataset.arff");
Instances data1 = source1.getDataSet();

if (data1.classIndex() == -1)
    data1.setClassIndex(data1.numAttributes() - 1);

double predicted=loadmod.classifyInstance(data1.instance(1));

System.out.println("Predicted label");
System.out.println(predicted);
}}
```

Note: To complete the program use curly braces at the end of the program one for closing the main class and another one for closing the main method as shown in the below screenshot of complete code.

Snapshot of complete code to develop and evaluate the Machine Learning Model from the dataset

```
package Performing_Prediction;
import java.io.File;
import weka.classifiers.trees.J48;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.misc.SerializedClassifier;
import weka.core.Debug;
import weka.core.Instances;

public class Performing_Prediction{
    public static void main(String[] args) throws Exception {

        DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
        Instances data = source.getDataSet();

        if (data.classIndex() == -1)
            data.setClassIndex(data.numAttributes() - 1);

        J48 tree = new J48();
        tree.buildClassifier(data);

        Debug.saveToFile("D:\\Ml_model\\tree_Ml_model.modle", tree);

        SerializedClassifier loadmod = new SerializedClassifier();
        loadmod.setModelFile(new File("D:\\Ml_model\\tree_Ml_model.modle"));

        DataSource source1 = new DataSource("D:\\Ml_model\\diabetes_Unknown_Dataset.arff");
        Instances data1 = source1.getDataSet();

        if (data1.classIndex() == -1)
            data1.setClassIndex(data1.numAttributes() - 1);

        double predicted=loadmod.classifyInstance(data1.instance(1));

        System.out.println("Predicted label");
        System.out.println(predicted);
    }
}
```

```
package Performing_Prediction;
import java.io.File;
import weka.classifiers.trees.J48;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.misc.SerializedClassifier;
import weka.core.Debug;
import weka.core.Instances;

public class Performing_Prediction{
    public static void main(String[] args) throws Exception {

        DataSource source = new DataSource("C:\\\\Program Files\\\\Weka-3-
8\\\\data\\\\diabetes.arff");
        Instances data = source.getDataSet();

        if (data.classIndex() == -1)
            data.setClassIndex(data.numAttributes() - 1);

        J48 tree = new J48();
        tree.buildClassifier(data);

        Debug.saveToFile("D:\\Ml_model\\\\tree_Ml_model.modle",
tree);

        SerializedClassifier loadmod = new SerializedClassifier();
        loadmod.setModelFile(new
File("D:\\Ml_model\\\\tree_Ml_model.modle"));

        DataSource source1 = new
DataSource("D:\\Ml_model\\\\diabetes_Unknown_Dataset.arff");
        Instances data1 = source1.getDataSet();

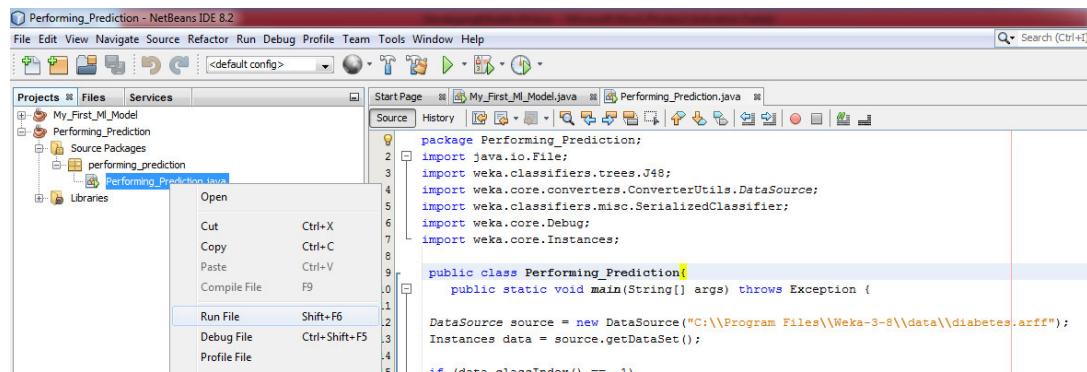
        if (data1.classIndex() == -1)
            data1.setClassIndex(data1.numAttributes() - 1);

        double predicted=loadmod.classifyInstance(data1.instance(1));

        System.out.println("Predicted label");
        System.out.println(predicted);
    }
}
```

Step VII: Run the code and display the results

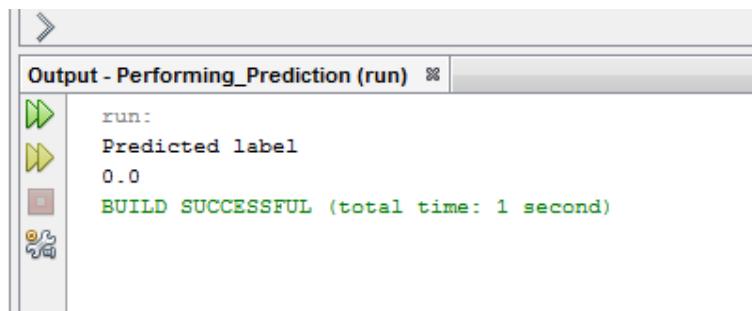
Do right-click on the Java class file named as “Performing_Prediction” one drop down menu is appeared and click on the “Run File” option to run the code and display the results as shown in the screenshot below. Or Simply Press the keys (Shift+F6) to run the code display the results.



The screenshot shows the NetBeans IDE interface. The title bar reads "Performing_Prediction - NetBeans IDE 8.2". The menu bar includes File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, Help. The search bar says "Search (Ctrl+F)". The Projects tab shows a project named "My_First_ML_Model" with a sub-project "Performing_Prediction". The Files tab shows a file named "My_First_ML_Model.java" and "Performing_Prediction.java". The Services tab is empty. The central pane displays the Java code for "Performing_Prediction.java". A context menu is open over the code editor, listing options like Cut (Ctrl+X), Copy (Ctrl+C), Paste (Ctrl+V), Compile File (F9), Run File (Shift+F6), Debug File (Ctrl+Shift+F5), and Profile File.

```
package Performing_Prediction;
import java.io.File;
import weka.classifiers.trees.J48;
import weka.core.converters.ConverterUtils.DataSource;
import weka.core.classifiers.misc.SerializedClassifier;
import weka.core.Debug;
import weka.core Instances;
public class Performing_Prediction{
    public static void main(String[] args) throws Exception {
        DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
        Instances data = source.getDataSet();
        if (data.classIndex() == -1)
```

Note: The results are appeared in the output window as shown in screenshot below



The screenshot shows the "Output - Performing_Prediction (run)" window in NetBeans. It displays the build log and the prediction result. The log shows "run:", "Predicted label", "0.0", and "BUILD SUCCESSFUL (total time: 1 second)".

```
run:
Predicted label
0.0
BUILD SUCCESSFUL (total time: 1 second)
```

Part II Additional Workouts

1. Develop a Multilayer Perceptron MI Model on the dataset weather.nominal and perform the display the model.

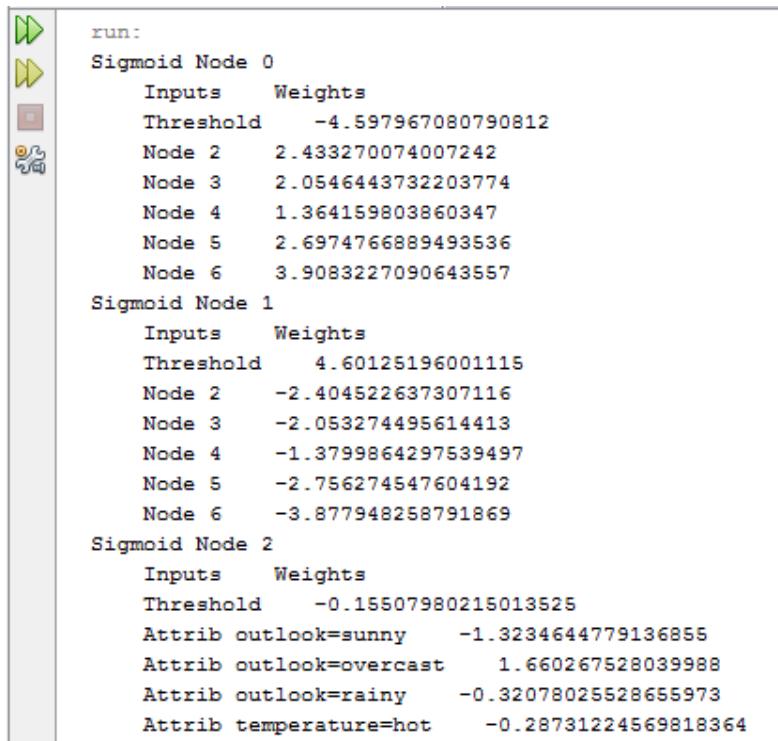
Screenshot of code:

```
DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\weather.nominal.arff");
Instances data = source.getDataSet();
    //set the class attribute
if (data.classIndex() == -1)
    data.setClassIndex(data.numAttributes() - 1);
//develop model
MultilayerPerceptron mlp = new MultilayerPerceptron();
mlp.buildClassifier(data);

System.out.println(mlp);

}}
```

Screenshot of output:



```
run:
Sigmoid Node 0
Inputs      Weights
Threshold   -4.597967080790812
Node 2      2.433270074007242
Node 3      2.0546443732203774
Node 4      1.364159803860347
Node 5      2.6974766889493536
Node 6      3.9083227090643557
Sigmoid Node 1
Inputs      Weights
Threshold   4.601251960011115
Node 2      -2.404522637307116
Node 3      -2.053274495614413
Node 4      -1.3799864297539497
Node 5      -2.756274547604192
Node 6      -3.877948258791869
Sigmoid Node 2
Inputs      Weights
Threshold   -0.15507980215013525
Attrib outlook=sunny   -1.3234644779136855
Attrib outlook=overcast  1.660267528039988
Attrib outlook=rainy    -0.32078025528655973
Attrib temperature=hot   -0.28731224569818364
```

Explanation:

Definition:	MultilayerPerceptron uses backpropagation to learn a multi-layer perceptron to classify instances buildClassifier method generates the classifier (Machine learning model) Note: mlp is the object of the class MultilayerPerceptron and the data is the object of the class DataSource System.out.println(mlp); displays value model MLP
Code:	MultilayerPerceptron mlp = new MultilayerPerceptron(); mlp.buildClassifier(data); System.out.println(mlp);
Ref. URL	http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/MultilayerPerceptron.html

- b. Develop a multilayer perceptron MI Model on the dataset weather.nominal and perform prediction on the 5 unknown instances and display their class label.**

```

1  @relation weather.symbolic
2
3  @attribute outlook {sunny, overcast, rainy}
4  @attribute temperature {hot, mild, cool}
5  @attribute humidity {high, normal}
6  @attribute windy {TRUE, FALSE}
7  @attribute play {yes, no}
8
9  @data
10 sunny,hot,high,FALSE,?
11 sunny,hot,high,TRUE,?
12 overcast,hot,high,FALSE,?
13 rainy,mild,high,FALSE,?
14 rainy,cool,normal,FALSE,?

```

Screenshot of code:

```

//load unknown dataset
DataSource source1 = new DataSource("D:\\ML_model\\weather.nominal_unknown.arff");
Instances data1 = source1.getDataSet();

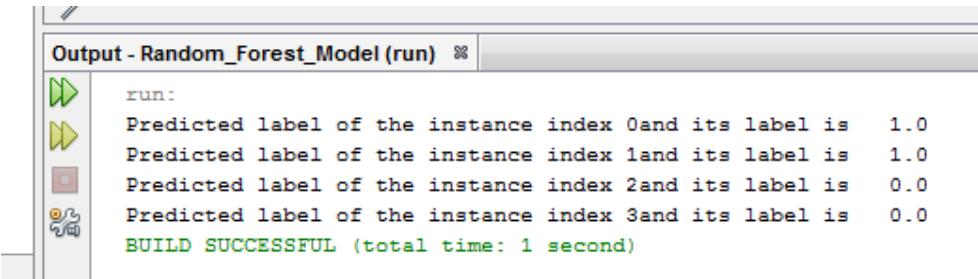
//set class attribute for unknown dataset
if (data1.classIndex() == -1)
    data1.setClassIndex(data1.numAttributes() - 1);

for(int i=0;i<4;i++)
{
    //perform prediction

    double predicted=loadmod.classifyInstance(data1.instance(i));
    //display predicted label
    System.out.println("Predicted label of the instance index "+i + "and its label is   "+predicted);
}

```

Screenshot of output:



```
run:  
Predicted label of the instance index 0 and its label is 1.0  
Predicted label of the instance index 1 and its label is 1.0  
Predicted label of the instance index 2 and its label is 0.0  
Predicted label of the instance index 3 and its label is 0.0  
BUILD SUCCESSFUL (total time: 1 second)
```

Explanation:

Definition:	for statement provides a way to iterate over a range of values. <code>classifyInstance(Instance instance)</code> classifies the given unknown instance.
Code:	<pre>for(int i=0;i<4;i++) { double predicted=loadmod.classifyInstance(data1.instance(i)); System.out.println("Predicted label of the instance index " + i + " and its label is " + predicted);</pre>
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/core/Instances.html https://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html

Chapter 4 Performing feature selection

Part I Performing feature selection

Outline of the program

The steps to developed the java program is shown in the below Figure. Initially, the required packages are imported. The significant features are selected from the loaded training dataset. Machine learning model is developed from the selected feature from the training dataset. The performances are evaluated on the developed model and the results are displayed.

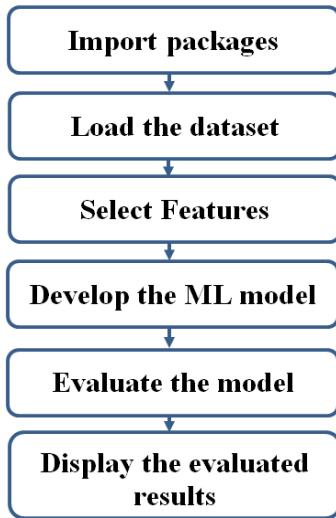


Figure: Steps to develop a Java code for performing feature selection

Step I: Develop the Java code to construct and predict label of the unknown data MI models

1. Import the required packages:

a. Import Package to develop the MI model (J48)

Definition:	J48 class is used to generate decision tree-based machine learning model.
Code:	<code>import weka.classifiers.trees.J48;</code>
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html

b. Import package to access the dataset from a file

Definition:	DataSource class is used to load the data from files
Code:	<code>Import weka.core.converters.ConverterUtils.DataSource;</code>
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/core/converters/ConverterUtil s.DataSource.html

c. Import package to evaluating the performance of the MI model

Definition	Evaluation class for evaluating machine learning models
Code:	import weka.classifiers.Evaluation;
Ref. URL	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

d. Import package to generate the random number

Definition:	Random class is used to generate a stream of pseudorandom numbers
Code:	import java.util.Random;
Ref. URL:	https://docs.oracle.com/javase/8/docs/api/java/util/Random.html

e. Import package to handle the instance of the dataset

Definition:	Instances class is used to handle the ordered set of weighted instances.
Code:	import weka.core.Instances;
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/core/Instances.html

f. Import package to select the correlated features

Definition:	CfsSubsetEval forms subsets of features that are highly correlated with the class while having low intercorrelation are preferred.
Code:	import weka.attributeSelection.CfsSubsetEval;
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/attributeSelection/CfsSubsetEval.html

g. Import package to handle the instance of the dataset

Definition:	GreedyStepwise performs a greedy forward or backward search through the space of attribute subsets
Code:	import weka.attributeSelection.GreedyStepwise;
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/attributeSelection/GreedyStepwise.html

h. Import package to select the correlated features

Definition:	AttributeSelectedClassifier used to reduce the dimensionality of training dataset by attribute selection before being passed on to a classifier.
Code:	import weka.classifiers.meta.AttributeSelectedClassifier;
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/meta/AttributeSelectedClassifier.html

Snapshot of code to import required classes:

```
import weka.classifiers.trees.J48;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.Evaluation;
import java.util.Random;
import weka.core.Instances;
import weka.attributeSelection.CfsSubsetEval;
import weka.attributeSelection.GreedyStepwise;
import weka.classifiers.meta.AttributeSelectedClassifier;
```

```
1 package Performing_Feature_Selection;
2 import weka.classifiers.trees.J48;
3 import weka.core.converters.DataSource;
4 import weka.classifiers.Evaluation;
5 import java.util.Random;
6 import weka.core.Instances;
7 import weka.attributeSelection.CfsSubsetEval;
8 import weka.attributeSelection.GreedyStepwise;
9 import weka.classifiers.meta.AttributeSelectedClassifier;
10
```

Step II: Define the Main class and the Main method

Note: Below a Java code is provided for displaying the string "HelloWorld"

```
public class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the string.
    }
}
```

- public class: Any other class can access a public field or method.
- public method: This method is public and therefore available to anyone.
- static method: This method can be run without having to create an instance of the class MyClass.
- void method: This method does not return anything.
- (String[] args): This method takes a String argument. Note that the argument args can be anything — it's common to use "args" but we could instead call it "stringArray".

Ref. URL: <https://docs.oracle.com/javase/tutorial/getStarted/application/index.html>

Definition:	public class: Any other class can access a public field or method. public method: This method is public and therefore available to anyone. static method: This method can be run without having to create an instance of the class MyClass. void method: This method does not return anything. (String[] args): This method takes a String argument. Note that the argument args can be anything — it's common to use "args" but we could instead call it "stringArray". The throws keyword indicates what exception type may be thrown by a method.
Code:	public class Performing_Prediction{ public static void main(String[] args) throws Exception {
Ref. URL:	https://docs.oracle.com/javase/tutorial/getStarted/application/index.html

Snapshot of code to define the Main class and the Main method

```
public class Performing_Prediction{
    public static void main(String[] args) throws Exception {
```

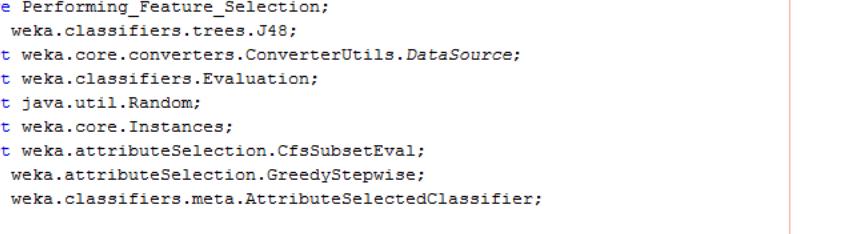
```
1 package Performing_Feature_Selection;
2 import weka.classifiers.trees.J48;
3 import weka.core.converters.ConverterUtils.DataSource;
4 import weka.classifiers.Evaluation;
5 import java.util.Random;
6 import weka.core.Instances;
7 import weka.attributeSelection.CfsSubsetEval;
8 import weka.attributeSelection.GreedyStepwise;
9 import weka.classifiers.meta.AttributeSelectedClassifier;
0
1 public class Performing_Feature_Selection {
2     public static void main(String[] args) throws Exception {
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
179
180
181
182
183
184
185
186
187
188
189
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
209
210
211
212
213
214
215
216
217
218
219
219
220
221
222
223
224
225
226
227
227
228
229
229
230
231
232
233
234
235
236
237
237
238
239
239
240
241
242
243
244
245
245
246
247
247
248
248
249
249
250
251
251
252
252
253
253
254
254
255
255
256
256
257
257
258
258
259
259
260
260
261
261
262
262
263
263
264
264
265
265
266
266
267
267
268
268
269
269
270
270
271
271
272
272
273
273
274
274
275
275
276
276
277
277
278
278
279
279
280
280
281
281
282
282
283
283
284
284
285
285
286
286
287
287
288
288
289
289
290
290
291
291
292
292
293
293
294
294
295
295
296
296
297
297
298
298
299
299
300
300
301
301
302
302
303
303
304
304
305
305
306
306
307
307
308
308
309
309
310
310
311
311
312
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
```

Step III: Load the dataset from the file

	<p><code>DataSource</code> class is used to loading data from files <code>getDataSet()</code> method returns the full dataset, can be null in case of an error.</p>
Definition:	<p>In this code, an object <code>source</code> is created for the class <code>DataSource</code> and the object <code>source</code> is used to invoke the method <code>getDataSet()</code> using dot operator</p> <p>Note: Provide the entire path of the dataset which should be loaded</p>
Code:	<pre>DataSource source = new DataSource("C:\\\\Program Files\\\\Weka-3-8\\\\data\\\\diabetes.arff"); Instances data = source.getDataSet();</pre>
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/core/converter/ConverterUtils.DataSource.html

Snapshot of code to load the full dataset

```
DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();
```



The screenshot shows a Java IDE interface with multiple tabs at the top: "My_First_ML_Model.java", "Performing_Prediction.java", "Random_Forest_Model.java", and "Performing_Feature_Selection.java...". The "Performing_Feature_Selection.java..." tab is active. Below the tabs is a toolbar with icons for file operations like new, open, save, and search. The main area displays the Java code for performing feature selection. The code imports various Weka classes and defines a main method that reads data from a local file named "diabetes.arff".

```
package Performing_Feature_Selection;
import weka.classifiers.trees.J48;
import weka.core.converters.DataSource;
import weka.classifiers.Evaluation;
import java.util.Random;
import weka.core.Instances;
import weka.attributeSelection.CfsSubsetEval;
import weka.attributeSelection.GreedyStepwise;
import weka.classifiers.meta.AttributeSelectedClassifier;

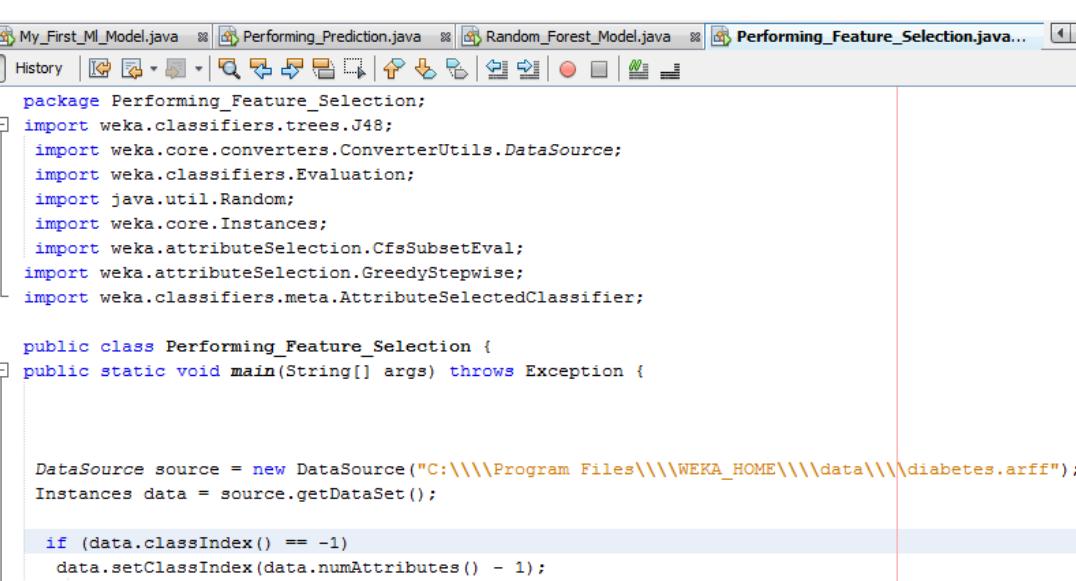
public class Performing_Feature_Selection {
    public static void main(String[] args) throws Exception {
        DataSource source = new DataSource("C:\\\\Program Files\\\\WEKA_HOME\\\\data\\\\diabetes.arff");
        Instances data = source.getDataSet();
```

Step IV: Set the class attribute for the dataset

Definition:	<code>classIndex()</code> Returns the class attribute's index (default it is set -1). <code>setClassIndex(int classIndex)</code> Sets the class index of the dataset.
--------------------	---

	<p>numAttributes () Returns the total number of attributes of the dataset</p> <p>Note: all the above methods are invoked by the object <code>data</code> which is created for the class <code>Instances</code>.</p>
Code:	<pre>if (data.classIndex() == -1) data.setClassIndex(data.numAttributes() - 1);</pre>
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/core/convertisers/ConverterUtils.DataSource.html

Snapshot of code to set the class attribute for the dataset



```

if (data.classIndex() == -1)
    data.setClassIndex(data.numAttributes() - 1);

package Performing_Feature_Selection;
import weka.classifiers.trees.J48;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.Evaluation;
import java.util.Random;
import weka.core.Instances;
import weka.attributeSelection.CfsSubsetEval;
import weka.attributeSelection.GreedyStepwise;
import weka.classifiers.meta.AttributeSelectedClassifier;

public class Performing_Feature_Selection {
public static void main(String[] args) throws Exception {

    DataSource source = new DataSource("C:\\\\Program Files\\\\WEKA_HOME\\\\data\\\\diabetes.arff");
    Instances data = source.getDataSet();

    if (data.classIndex() == -1)
        data.setClassIndex(data.numAttributes() - 1);
}
}

```

Step V: Perform feature selection

Definition:	<p><code>AttributeSelectedClassifier</code> used to reduce the dimensionality of training dataset by attribute selection before being passed on to a classifier.</p> <p><code>CfsSubsetEval</code> forms subsets of features that are highly correlated with the class while having low intercorrelation are preferred.</p> <p><code>GreedyStepwise</code> performs a greedy forward or backward search through the space of attribute subsets</p> <p><code>setSearchBackwards(boolean back)</code> used to set whether to search backwards instead of forwards</p>
Code:	<pre>AttributeSelectedClassifier classifier = new AttributeSelectedClassifier(); CfsSubsetEval eval = new CfsSubsetEval(); GreedyStepwise search = new GreedyStepwise(); search.setSearchBackwards(true);</pre>
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/meta/AttributeSelectedClassifier.html http://weka.sourceforge.net/doc.dev/weka/attributeSelection/CfsSubsetEval.html http://weka.sourceforge.net/doc.dev/weka/attributeSelection/GreedyStepwise.html

Snapshot of code to develop the Machine Learning Model from the dataset

```

AttributeSelectedClassifier classifier = new AttributeSelectedClassifier();
CfsSubsetEval eval = new CfsSubsetEval();
GreedyStepwise search = new GreedyStepwise();
search.setSearchBackwards(true);

DataSource source = new DataSource("C:\\\\Program Files\\\\WEKA_HOME\\\\data\\\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
    data.setClassIndex(data.numAttributes() - 1);

AttributeSelectedClassifier classifier = new AttributeSelectedClassifier();
CfsSubsetEval eval = new CfsSubsetEval();
GreedyStepwise search = new GreedyStepwise();
search.setSearchBackwards(true);

```

Step V: Develop the machine learning model from the selected features dataset

Definition:	J48 class for developing a pruned or unpruned C4.5 decision tree buildClassifier method generates the classifier (Machine learning model) j48 is the object of the class J48 setClassifier(Classifier newClassifier) sets the base learner. setEvaluator(ASEvaluation evaluator) sets the attribute/subset evaluator setSearch(ASSearch search) sets the search method
Code:	J48 j48 = new J48(); classifier.setClassifier(j48); classifier.setEvaluator(eval); classifier.setSearch(search);
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html http://weka.sourceforge.net/doc.dev/weka/classifiers/SingleClassifierEnhancer.html http://weka.sourceforge.net/doc.dev/weka/attributeSelection/AttributeSelection.html

Snapshot of code to develop the Machine Learning Model from the dataset

```

J48 j48 = new J48();
classifier.setClassifier(j48);
classifier.setEvaluator(eval);
classifier.setSearch(search);

DataSource source = new DataSource("C:\\\\Program Files\\\\WEKA_HOME\\\\data\\\\diabetes.arff")
Instances data = source.getDataSet();

if (data.classIndex() == -1)
    data.setClassIndex(data.numAttributes() - 1);

```

```

21 AttributeSelectedClassifier classifier = new AttributeSelectedClassifier();
22 CfsSubsetEval eval = new CfsSubsetEval();
23 GreedyStepwise search = new GreedyStepwise();
24 search.setSearchBackwards(true);
25
26 J48 j48 = new J48();
27 classifier.setClassifier(j48);
28 classifier.setEvaluator(eval);
29 classifier.setSearch(search);

```

Step VI: Evaluate the performance of machine learning model on the selected feature set

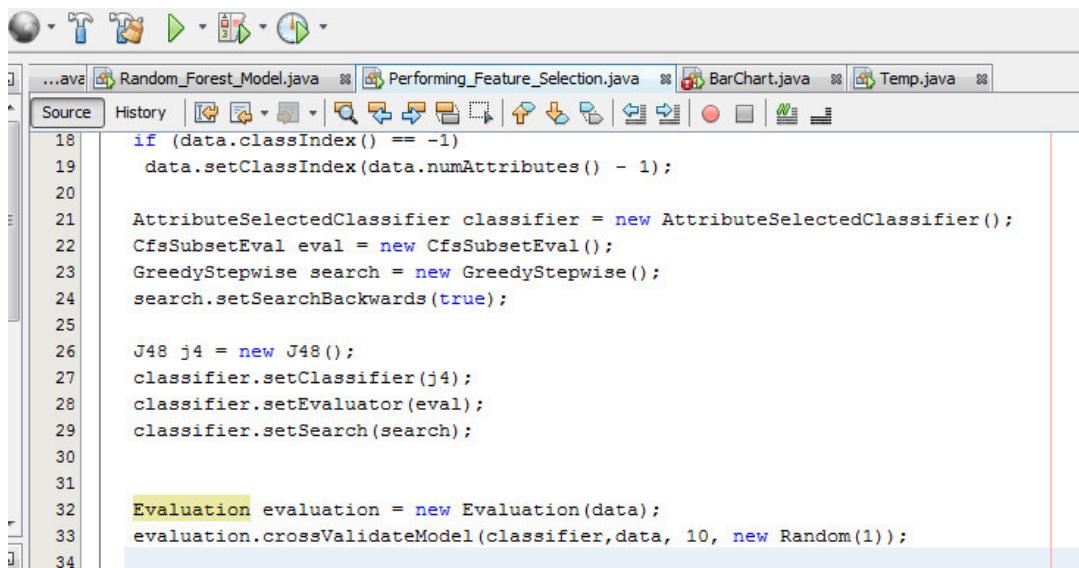
Definition	Evaluation class is used for evaluating machine learning models. crossValidateModel method Performs a (stratified if class is nominal) cross-validation for a classifier on a set of instances. Random class is used to generate a stream of pseudorandom numbers
Code:	Evaluation evaluation = new Evaluation(data); evaluation.crossValidateModel(classifier,data, 10, new Random(1));
Ref. URL	http://weka.sourceforge.net/doc.stable/weka/classifiers/Evaluation.html https://docs.oracle.com/javase/8/docs/api/java/util/Random.html

Snapshot of code to evaluate the performance of Machine learning model and display the result in console window

```

Evaluation evaluation = new Evaluation(data);
evaluation.crossValidateModel(classifier,data, 10, new Random(1));

```



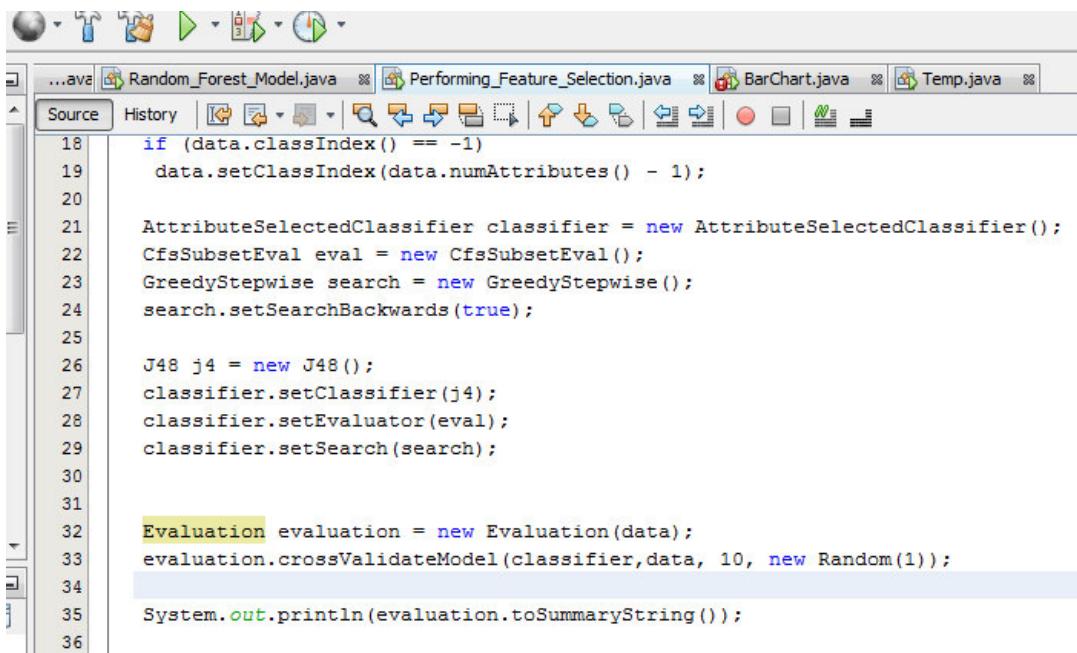
Step VI: Display the evaluated result

Definition:	System.out.println(data) is standard output stream that displays value of the data for output toSummaryString method outputs the performance statistics in summary form.
Code:	System.out.println(evaluation.toSummaryString());

Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/classifiers/Evaluation.html https://docs.oracle.com/javase/8/docs/api/java/lang/System.html
------------------	--

Snapshot of code to evaluate the performance of Machine learning model and display the result in console window

```
System.out.println(evaluation.toSummaryString());
```



The screenshot shows a Java code editor with several tabs at the top: "...java", "Random_Forest_Model.java", "Performing_Feature_Selection.java", "BarChart.java", and "Temp.java". The code itself is as follows:

```
18     if (data.classIndex() == -1)
19         data.setClassIndex(data.numAttributes() - 1);
20
21     AttributeSelectedClassifier classifier = new AttributeSelectedClassifier();
22     CfsSubsetEval eval = new CfsSubsetEval();
23     GreedyStepwise search = new GreedyStepwise();
24     search.setSearchBackwards(true);
25
26     J48 j4 = new J48();
27     classifier.setClassifier(j4);
28     classifier.setEvaluator(eval);
29     classifier.setSearch(search);
30
31
32     Evaluation evaluation = new Evaluation(data);
33     evaluation.crossValidateModel(classifier, data, 10, new Random(1));
34
35     System.out.println(evaluation.toSummaryString());
36
```

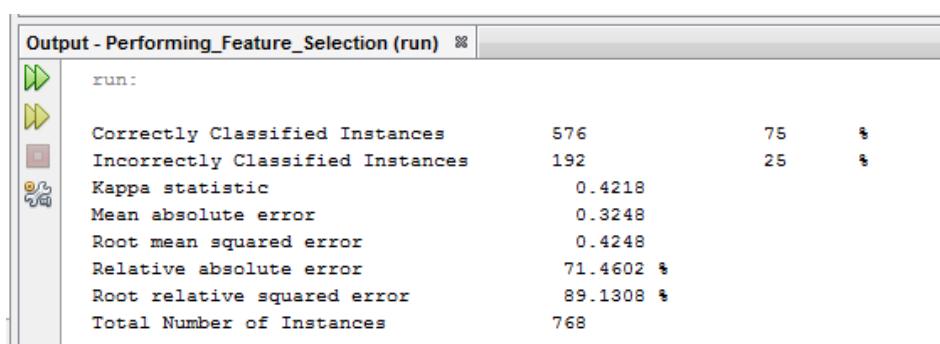
Note: To complete the program use curly braces at the end of the program one for closing the main class and another one for closing the main method as shown in the below screenshot of complete code.

```
Evaluation evaluation = new Evaluation(data);
evaluation.crossValidateModel(classifier, data, 10, new Random(1));

System.out.println(evaluation.toSummaryString());

}
}
```

Screenshot of output with feature selection:



The screenshot shows the "Output" window of an IDE with the title "Output - Performing_Feature_Selection (run)". The output text is as follows:

```
run:

Correctly Classified Instances      576      75 %
Incorrectly Classified Instances   192      25 %
Kappa statistic                   0.4218
Mean absolute error               0.3248
Root mean squared error          0.4248
Relative absolute error           71.4602 %
Root relative squared error      89.1308 %
Total Number of Instances        768
```

Snapshot of complete code to perform the feature selection

```
import weka.classifiers.trees.J48;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.Evaluation;
import java.util.Random;
import weka.core.Instances;
import weka.attributeSelection.CfsSubsetEval;
import weka.attributeSelection.GreedyStepwise;
import weka.classifiers.meta.AttributeSelectedClassifier;

public class Performing_Feature_Selection {
public static void main(String[] args) throws Exception {

    DataSource source = new DataSource("C:\\\\Program Files\\\\WEKA_HOME\\\\data\\\\diabetes.arff");
    Instances data = source.getDataSet();

    if (data.classIndex() == -1)
        data.setClassIndex(data.numAttributes() - 1);

    AttributeSelectedClassifier classifier = new AttributeSelectedClassifier();
    CfsSubsetEval eval = new CfsSubsetEval();
    GreedyStepwise search = new GreedyStepwise();
    search.setSearchBackwards(true);

    J48 j48 = new J48();
    classifier.setClassifier(j48);
    classifier.setEvaluator(eval);
    classifier.setSearch(search);

    Evaluation evaluation = new Evaluation(data);
    evaluation.crossValidateModel(classifier, data, 10, new Random(1));

    System.out.println(evaluation.toSummaryString());
}

}

import weka.classifiers.trees.J48;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.Evaluation;
import java.util.Random;
import weka.core.Instances;
import weka.attributeSelection.CfsSubsetEval;
import weka.attributeSelection.GreedyStepwise;
import weka.classifiers.meta.AttributeSelectedClassifier;

public class Performing_Feature_Selection {
public static void main(String[] args) throws Exception {

DataSource source = new DataSource("C:\\\\Program
Files\\\\WEKA_HOME\\\\data\\\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
    data.setClassIndex(data.numAttributes() - 1);

AttributeSelectedClassifier classifier = new
AttributeSelectedClassifier();
CfsSubsetEval eval = new CfsSubsetEval();
GreedyStepwise search = new GreedyStepwise();
search.setSearchBackwards(true);
```

```
J48 j48 = new J48();
classifier.setClassifier(j48);
classifier.setEvaluator(eval);
classifier.setSearch(search);

Evaluation evaluation = new Evaluation(data);
evaluation.crossValidateModel(classifier, data, 10, new
Random(1));

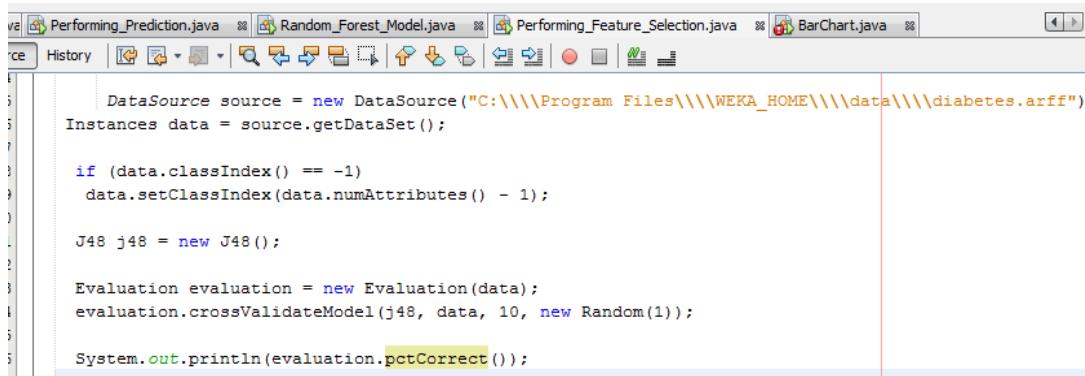
System.out.println(evaluation.toSummaryString());

}
```

Part II Additional Workouts

1. Develop a J48 model on diabetes dataset and compare the accuracy of the model with and without feature selection

Screenshot of code without feature selection:



```
DataSource source = new DataSource("C:\\\\Program Files\\\\WEKA_HOME\\\\data\\\\diabetes.arff")
Instances data = source.getDataSet();

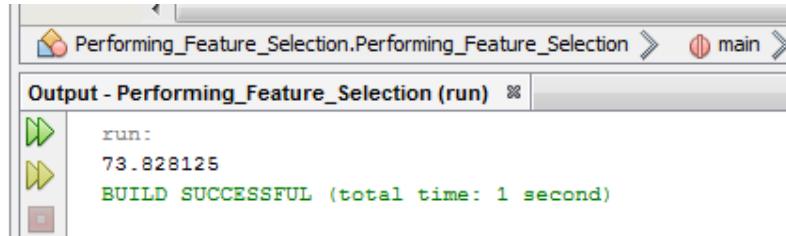
if (data.classIndex() == -1)
    data.setClassIndex(data.numAttributes() - 1);

J48 j48 = new J48();

Evaluation evaluation = new Evaluation(data);
evaluation.crossValidateModel(j48, data, 10, new Random(1));

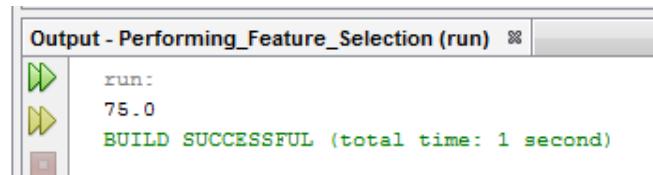
System.out.println(evaluation.pctCorrect());
```

Screenshot of output without feature selection:



```
run:
73.828125
BUILD SUCCESSFUL (total time: 1 second)
```

Screenshot of code with feature selection:



```
run:
75.0
BUILD SUCCESSFUL (total time: 1 second)
```

Comparison table for accuracy with and without feature selection

Accuracy in percentage without feature selection	Accuracy in percentage with feature selection
73.82	75.00

*Gaining knowledge,
is the first step to wisdom.*

*Sharing it,
is the first step to humanity.*