

Weka Based Desktop Data Mining as Web Service

Sujala.D.Shetty, S.Vadivel, Sakshi Vaghella

Abstract—Data mining is the process of sifting through large volumes of data, analyzing data from different perspectives and summarizing it into useful information. One of the widely used desktop applications for data mining is the Weka tool which is nothing but a collection of machine learning algorithms implemented in Java and open sourced under the General Public License (GPL). A web service is a software system designed to support interoperable machine to machine interaction over a network using SOAP messages. Unlike a desktop application, a web service is easy to upgrade, deliver and access and does not occupy any memory on the system. Keeping in mind the advantages of a web service over a desktop application, in this paper we are demonstrating how this Java based desktop data mining application can be implemented as a web service to support data mining across the internet.

Keywords—desktop application, Weka mining, web service

I. INTRODUCTION

THE open source code available on downloading the Weka 3.6 serves as the basis for us to convert the desktop application into a web service. Netbeans IDE 6 has been used to create the web service. We have taken as case studies three data mining tasks and for each of these tasks we have selected a data mining algorithm. In step 1 we have made use of libraries available in the open source code of Weka and used them in our own java code of data mining. Next in step 2 we have converted each of these codes of data mining into web services and correspondingly made clients for each of these three applications.

II. OVERVIEW OF DATA MINING

Data can be any fact, number, or text which can be processed by computer. Data mining uses sophisticated algorithms to extract useful data and identify trends and relationships in data that are beyond simple analysis. In data mining, there are several fundamental functions and associated algorithms. These mining functions include classification, regression, clustering and association and form a core supporting many common data mining solutions [1]. The technology of data mining is not new. It finds wide use in various domains like-

- Science and engineering: Changes in DNA sequence for detecting diseases [2].

- condition monitoring of high voltage electrical equipments.
- Dissolved Gas Analysis (DGA) on power transformers.

Business:

- Customer Relationship Management (CRM) to find prospects with high likelihood of responding.
- Market Basket Analysis for retail sales [3].

Mail Mining:

- Parse and store e-mails by senders/subject.
- Separate spam mail

Building a web portal (like MyYahoo):

- Collect data from more than one source page and present it as a single page by adding some value to the data.

Building a search engine (e.g. Google):

- Data mining helps Web search engines find high quality Web pages and enhances Web click stream analysis.
- Web search engine crawls the Web, indexes Web pages and builds and stores huge keyword based indices that help to locate sets of Web pages to a query.
- Return relevant pages to the query [3].

Personalized B2C E-Commerce (e.g. Amazon.com):

- A host of Web mining techniques, e.g. associations between pages visited, click-path analysis, wish lists, instant recommendations etc., are used to improve the customer's experience during a 'store visit'
- Also with the help of data mining techniques, the various patterns of customer web-usage can be found.
- Depending on these usage patterns, we can classify and categorize users and accordingly send promotions and discounts to the appropriate user groups[3].

Sujala D Shetty is working as Senior Lecturer (CS) at BITS, Pilani – Dubai, Dubai, U.A.E. e-mail: sujala@bitsdubai.com.

Dr. S.Vadivel is working as Associate Professor (CS) and Head of Department at BITS,Pilani- Dubai, Dubai, U.A.E. e-mail: vadivel@bitsdubai.com.

Sakshi Vaghella was a final year Computer Science student at BITS, Pilani – Dubai, Dubai, U.A.E.

III. WEKA FOR DATA MINING

Weka (Waikato Environment for Knowledge Analysis) is an ensemble of data mining algorithms written in Java. These algorithms can either be applied directly to a dataset using the Weka explorer or called from your own modified Java code. It contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization and can be used to develop new machine learning schemes [4].

Weka provides different environments like the Simple CLI (command-line interface) for direct execution of Weka commands, Explorer for exploring data with Weka and Experimenter for performing experiments and conducting statistical tests between learning schemes [5]. It is a desktop application and needs to be downloaded from the official web site. It requires that a specific Java version be installed on the system and is supported by Windows, MAC and Linux operating systems. The datasets for Weka should also be saved on the system and formatted according to the ARFF format. Converters included in WEKA can convert other file formats like CSV to ARFF [6].

IV. NEED FOR WEB SERVICE

So far we have had data mining applications which are run from the desktop. The user has to install the data mining engine in Oracle or the data mining software like WEKA or RAPID MINER and then solve the data mining tasks. This not only occupies a lot of memory on the system for storing the data repository and the software but is also not easily accessible from a system other than the system on which the software is installed. Also huge volumes of data available across the net cannot be mined successfully from the desktop. Web services are easy and inexpensive to deliver, upgrade and access. So if we convert a data mining application into a web application, the data mining application can be hosted and accessed from any client in the world. Any modifications can be made to the code without the need of downloading the updated code. Also the data on the net can be mined without the need to download the data, only the file URL needs to be sent to the web service. Then depending on the algorithm used in the web enabled data mining application, the application will return the results of the mined data. It also helps in easy interaction of the web service with large corporations who need to mine their data and get back the result anytime anywhere [7].

V. WEKA WEB SERVICE CREATION

Here we have made use of the libraries used in the open source code of Weka to implement certain data mining algorithms of J48 classification, EM clustering and also text classification as web services and created clients for all these three applications.

A. Convert J48 Classifier into Web Service

The notion of classification is to classify cases according to a fixed set of categories. In simple words, classification is a machine learning (data mining) technique to predict group

membership of instances. Decision trees represent a supervised approach to classification. A decision tree is a simple structure where non-terminal nodes represent tests on one or more attributes and terminal nodes reflect decision outcomes [8].

The J48 Decision tree classifier follows a simple algorithm. In order to classify a new item, it first needs to create a decision tree based on the attribute values of the available training data. So, whenever it encounters a set of items (training set) it identifies the attribute that discriminates the various instances most clearly [9].

The general approach can be summarized as given below:

1. Choose an attribute that best differentiates the output attribute values.
2. Create a separate tree branch for each value of the chosen attribute.
3. Divide the instances into subgroups so as to reflect the attribute values of the chosen node.
4. For each subgroup, terminate the attribute selection process if:
 - a. All members of a subgroup have the same value for the output attribute, terminate the attribute selection process for the current path and label the branch on the current path with the specified value.
 - b. The subgroup contains a single node or no further distinguishing attributes can be determined. As in (a), label the branch with the output value seen by the majority of remaining instances.
5. For each subgroup created in (3) that has not been labeled as terminal, repeat the above process.

This algorithm is applied to the training data. The code for the above as a web service with the necessary comments is as shown below. The web service makes use of the method 'execute' which takes as an input a string parameter specifying the name of classifier algorithm, the filter and also the file location of the dataset. It then trains the iris dataset using the J48 decision tree algorithm and outputs the results as shown in Listing 1.

```
/*
 * To change this template, choose Tools / Templates
 * and open the template in the editor.
 */
```

```
package demo1;
```

```
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import weka.classifiers.Classifier;
import weka.classifiers.Evaluation;
```

```

import weka.core.Instances;
import weka.core.OptionHandler;
import weka.core.Utils;
import weka.filters.Filter;

import java.io.FileReader;
import java.io.BufferedReader;
import java.util.Vector;

@WebService()
public class NewWebService12 {

    /**
     * Web service operation
     */

    @WebMethod(operationName = "execute")
    public String execute(@WebParam(name = "input")
        String input)throws Exception {

        Classifier m_Classifier = null;

        /** the filter to use */
        Filter m_Filter = null;

        /** the training file */
        String m_TrainingFile = null;

        /** the training instances */
        Instances m_Training = null;

        /** for evaluating the classifier */
        Evaluation m_Evaluation = null;

        String classifier = "";
        String filter = "";
        String dataset = "";
        Vector classifierOptions = new Vector();
        Vector filterOptions = new Vector();

        System.out.println(input);

        /**splitting the input into six parts */
        String args[] = input.split("\\s");
        for ( int i=0;i < args.length;i++ )
            System.out.println ( args [ i ] );

        int i = 0;
        String current = "";
        boolean newPart = false;
        do {

            // determine part of command line

            if (args[i].equals("CLASSIFIER")) {
                System.out.println(args[i]);
                current = args[i];
                i++;
                newPart = true;
            }
            else if (args[i].equals("FILTER")) {
                current = args[i];
                i++;
                newPart = true;
            }
            else if (args[i].equals("DATASET")) {
                current = args[i];
                i++;
                newPart = true;
            }

            if (current.equals("CLASSIFIER")) {
                if (newPart)
                    classifier = args[i];
                else
                    classifierOptions.add(args[i]);
            }
            else if (current.equals("FILTER")) {
                if (newPart)
                    filter = args[i];
                else
                    filterOptions.add(args[i]);
            }
            else if (current.equals("DATASET")) {
                if (newPart)
                    dataset = args[i];

                //dataset="C:/Users/s1/Documents/NetBeansProjects/JavaA
                pp/build/classes/iris.arff ";
            }

            // next parameter
            i++;
            newPart = false;
        }
        while (i < args.length);

        m_Classifier = Classifier.forName(classifier,

            (String[]) classifierOptions.toArray(new
            String[classifierOptions.size()]));

        m_Filter = (Filter) Class.forName(filter).newInstance();
        if (m_Filter instanceof OptionHandler)
            ((OptionHandler) m_Filter).setOptions((String[])
            filterOptions.toArray(new String[filterOptions.size()]));

        m_TrainingFile = dataset;
        m_Training = new Instances(
            new BufferedReader(new
            FileReader(m_TrainingFile)));
        m_Training.setClassIndex(m_Training.numAttributes() -
            1);

```

```

// run filter
m_Filter.setInputFormat(m_Training);
Instances filtered = Filter.useFilter(m_Training, m_Filter);

// train classifier on complete file for tree
m_Classifier.buildClassifier(filtered);

// 10fold CV with seed=1
m_Evaluation = new Evaluation(filtered);
m_Evaluation.crossValidateModel(
    m_Classifier, filtered, 10,
m_Training.getRandomNumberGenerator(1));

StringBuffer    result;

result = new StringBuffer();
result.append("Weka - Demo\n=====\\n\\n");

result.append("Classifier...: "
    + m_Classifier.getClass().getName() + " "
    + Utils.joinOptions(m_Classifier.getOptions()) + "\\n");
if (m_Filter instanceof OptionHandler)
    result.append("Filter.....: "
        + m_Filter.getClass().getName() + " "
        + Utils.joinOptions(((OptionHandler)
m_Filter).getOptions()) + "\\n");
else
    result.append("Filter.....: "
        + m_Filter.getClass().getName() + "\\n");
result.append("Training file: "
    + m_TrainingFile + "\\n");
result.append("\\n");

result.append(m_Classifier.toString() + "\\n");
result.append(m_Evaluation.toSummaryString() + "\\n");
try {
    result.append(m_Evaluation.toMatrixString() + "\\n");
} catch (Exception e) {
    e.printStackTrace();
}
try {
    result.append(m_Evaluation.toClassDetailsString() +
"\\n");
}
catch (Exception e) {
    e.printStackTrace();
}

return result.toString();

}
}

```

Listing 1

B. Convert EM Clusterer into Web Service

Clustering analysis identifies clusters that exist in a given dataset, where a cluster is a collection of cases that are more similar to one another than cases in other clusters. A set of

clusters is considered to be of high quality if the similarity between clusters is low, yet the similarity of cases within a cluster is high [10].

The Expectation-Maximization (EM) algorithm is part of the Weka clustering package. It is a statistical model and makes use of the finite Gaussian mixtures model that assumes all attributes to be independent random variables. The parameters are re-computed until a desired convergence value is achieved [11].

In the simplest case, the probability distributes are assumed to be normal and data instances consist of a single real-valued attribute. Using the scenario, the job of the algorithm is to determine the value of five parameters, specifically:

- The mean and standard deviation for cluster 1
- The mean and standard deviation for cluster 2
- The sampling probability for cluster 1 (or the probability for cluster 2)

Here is the general procedure:

- Guess initial values for the five parameters.
- Use the probability density function for a normal distribution to compute the cluster probability for each instance. In the two-cluster case, we will have the two probability distribution formulas each having differing mean and standard deviation values.
- Use the probability scores to re-estimate the five parameters [12].

This algorithm is applied to the training data. The code for the above as a web service with the necessary comments is as shown below. Here the web service makes use of the method 'execute' whose input parameter is the file URL of the dataset. It then trains the weather dataset using the EM Clustering algorithm and outputs the results as shown in Listing 2.

```

package demo;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

import weka.core.Instances;
import weka.clusterers.DensityBasedClusterer;
import weka.clusterers.EM;
import weka.clusterers.ClusterEvaluation;

import java.io.FileReader;
import java.io.BufferedReader;

@WebService()
public class NewWebService2 {

    /**
     * Web service operation
     */

    @WebMethod(operationName = "execute")
    public String execute(@WebParam(name = "input")

```

```

final String input)throws Exception{

    ClusterEvaluation eval;
    Instances      data;
    String[]       options;
    DensityBasedClusterer cl;

    data = new Instances(new BufferedReader(new
    FileReader(input)));
    StringBuffer    result;

    result = new StringBuffer();

    result.append("Weka - Demo\n=====\\n\\n" + "\\n\\n-
-> normal\\n");

    // normal
    //System.out.println("\\n--> normal");

    options  = new String[2];
    options[0] = "-t";
    options[1] = input;

    //System.out.println(ClusterEvaluation.evaluateClusterer(new
    EM(), options));

    result.append(ClusterEvaluation.evaluateClusterer(new
    EM(), options));

    // manual call
    //System.out.println("\\n--> manual");

    cl = new EM();
    cl.buildClusterer(data);
    eval = new ClusterEvaluation();
    eval.setClusterer(cl);
    eval.evaluateClusterer(new Instances(data));

    //System.out.println("# of clusters: " +
    eval.getNumClusters());

    try {
        result.append("\\n--> manual" + "\\n\\n# of clusters: " +
        eval.getNumClusters() + "\\n");
    }
    catch (Exception e) {
        e.printStackTrace();
    }

    // density based
    //System.out.println("\\n--> density (CV)");
    cl = new EM();

    eval = new ClusterEvaluation();
    eval.setClusterer(cl);
    eval.crossValidateModel(
        cl, data, 10, data.getRandomNumberGenerator(1));

```

```

    //System.out.println("# of clusters: " +
    eval.getNumClusters());

    try {
        result.append("\\n--> density (CV)" + "\\n\\n# of clusters: " +
        eval.getNumClusters() + "\\n\\n");
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    return result.toString();
}
}

```

Listing 2

C. Convert Text Classifier into Web Service

The given code for training and testing a TextClassifier can be implemented with two algorithms – IBk Lazy and Naïve Bayes. This code can be extensively used for the purpose of mail mining and depending on the content of the mail it can be classified as SPAM or NO SPAM.

Here we have made use of the Lazy IBk which is Weka's implementation of the K – Nearest Neighbor Classification algorithm [8]. The k-nearest neighbors algorithm (k-NN) is a method for classifying objects based on closest training examples in the feature space. It is one of the simplest algorithms that supports a type of instance-based learning, or lazy learning where the function is only approximated locally and all computation is deferred until classification. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common amongst its k nearest neighbors. k is a positive integer, typically small. If k = 1, then the object is simply assigned to the class of its nearest neighbor. In binary (two class) classification problems, it is helpful to choose k to be an odd number as this avoids tied votes. The neighbors are taken from a set of objects for which the correct classification (or, in the case of regression, the value of the property) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required [13].

The code for the above as a web service for text classification with the necessary comments is as shown below. Here the web service makes use of the 'execute' method which does not have any input parameters. The training and testing dataset are stored as string arrays and are classified into 'spam and 'no spam' by making use of the Lazy IBk algorithm and the results are as shown in Listing 3.

```

package demo;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

import java.util.*;

import weka.core.*;

```

```

import weka.core.Instance;
import weka.core.Instances;
import weka.core.Attribute;

import weka.classifiers.*;
import weka.classifiers.Classifier;
import
weka.filters.unsupervised.attribute.StringToWordVector;

@WebService()
public class TextClassifierService {

    /**
     * Web service operation
     */
    @WebMethod(operationName = "execute")
    public String execute(@WebParam(name = "input")
        String input) {

        //TODO write your implementation code here:

        Instances instances    = null;
        Classifier classifier   = null;
        Instances filteredData  = null;
        Evaluation evaluation   = null;
        Set    modelWords      = null;

        // maybe this should be settable?

        String delimitersStringToWordVector = "\\s,,:\"\\\"()?!";

        // String classString =
"weka.classifiers.bayes.NaiveBayes";
String classString = "weka.classifiers.lazy.IBk";

        // String classString = input;

        String[] inputText = {"hey, buy this from me!", "do you want
        to buy?", "I have a party tonight!", "today it is a nice
        weather", "you are best", "I have a horse", "you are my
        friend", "buy, buy, buy!", "it is spring in the air", "do you want
        to come?"};

        String[] inputClasses = {"spam", "spam", "no spam", "no
        spam", "spam", "no spam", "no spam", "spam", "no spam", "no
        spam"};

        String[] testText = {"you want to buy from me?", "usually
        I run in stairs", "buy it now!", "buy, buy, buy!", "you are the
        best, buy!", "it is spring in the air"};

        if (inputText.length != inputClasses.length) {
            System.err.println("The length of text and classes must
            be the same!");
            System.exit(1);
        }
    }

```

```

// calculate the classValues

    HashSet classSet = new
    HashSet(Arrays.asList(inputClasses));
    classSet.add("");
    String[] classValues = (String[])classSet.toArray(new
    String[0]);

    //
// create class attribute
//
    FastVector classAttributeVector = new FastVector();
    for (int i = 0; i < classValues.length; i++) {
        classAttributeVector.addElement(classValues[i]);
    }
    Attribute ClassAttribute = new Attribute("class",
    classAttributeVector);

    //
// create text attribute
//
    FastVector inputTextVector = null; // null -> String type
    Attribute TextAttribute = new Attribute("text",
    inputTextVector);
    for (int i = 0; i < inputText.length; i++) {
        TextAttribute.addStringValue(inputText[i]);
    }

    // add the text of test cases
    for (int i = 0; i < testText.length; i++) {
        TextAttribute.addStringValue(testText[i]);
    }

    //
// create the attribute information
//
    FastVector AttributeInfo = new FastVector(2);
    AttributeInfo.addElement(TextAttribute);
    AttributeInfo.addElement(ClassAttribute);

    /*this.inputText    = inputText;
this.inputClasses   = inputClasses;
this.classString     = classString;
this.attributeInfo   = attributeInfo;
this.textAttribute   = textAttribute;
this.classAttribute  = classAttribute;
*/

    StringBuffer result = new StringBuffer();

    result.append("dataset:\n\n");

    // creates an empty instances set
    instances = new Instances("data set", AttributeInfo, 100);

    // set which attribute is the class attribute
    instances.setClass(ClassAttribute);

```

```

try {

    for (int i = 0; i < inputText.length; i++) {
        Instance inst = new Instance(2);
        inst.setValue(TextAttribute, inputText[i]);
        if (inputClasses != null && inputClasses.length > 0) {
            inst.setValue(ClassAttribute, inputClasses[i]);
        }
        instances.add(inst);
    }
    result.append("DATA SET:\n" + instances + "\n");

    StringToWordVector filter = null;

    // default values according to Java Doc:
    int wordsToKeep = 1000;

    Instances filtered = null;

    try {

        filter = new StringToWordVector(wordsToKeep);
        filter.setOutputWordCounts(true);
        filter.setSelectedRange("1");

        filter.setInputFormat(instances);

        filtered = weka.filters.Filter.useFilter(instances, filter);
        // System.out.println("filtered:\n" + filtered);

    } catch (Exception e) {
        e.printStackTrace();
    }
    filteredData = filtered;

    // create Set of modelWords

    modelWords = new HashSet();
    Enumeration enumx =
filteredData.enumerateAttributes();
    while (enumx.hasMoreElements()) {
        Attribute att = (Attribute)enumx.nextElement();
        String attName = att.name().toLowerCase();
        modelWords.add(attName);
    }

    classifier = Classifier.forName(classString, null);

    classifier.buildClassifier(filteredData);
    evaluation = new Evaluation(filteredData);
    evaluation.evaluateModel(classifier, filteredData);

    try {
        result.append("\n\nINFORMATION ABOUT THE
CLASSIFIER AND EVALUATION:\n");
        result.append("\nclassifier.toString():\n" +
classifier.toString() + "\n");

        result.append("\nevaluation.toSummaryString(title,
false):\n" + evaluation.toSummaryString("Summary", false) +
"\n");
        result.append("\nevaluation.toMatrixString():\n" +
evaluation.toMatrixString() + "\n");
        result.append("\nevaluation.toClassDetailsString():\n"
+ evaluation.toClassDetailsString("Details") + "\n");

        result.append("\nevaluation.toCumulativeMarginDistribution:\n"
+ evaluation.toCumulativeMarginDistributionString() +
"\n");
    } catch (Exception e) {
        e.printStackTrace();
        result.append("\nException (sorry!):\n" + e.toString());
    }

    result.append("\n\n");

    // check instances
    int startIx = 0;
    String testType = "not test";

    try {

        result.append("\nCHECKING ALL THE
INSTANCES:\n");

        Enumeration enumClasses =
ClassAttribute.enumerateValues();
        result.append("Class values (in order): ");
        while (enumClasses.hasMoreElements()) {
            String classStr =
(String)enumClasses.nextElement();
            result.append("'" + classStr + "' ");
        }
        result.append("\n");

        // startIx is a fix for handling text cases
        for (int i = startIx; i < filteredData.numInstances();
i++) {

            SparseInstance sparseInst = new
SparseInstance(filteredData.instance(i));
            sparseInst.setDataset(filteredData);

            result.append("\nTesting: '" + inputText[i-startIx] +
"\n");
            // result.append("SparseInst: " + sparseInst +
"\n");

            double correctValue =
(double)sparseInst.classValue();
            double predictedValue =
classifier.classifyInstance(sparseInst);

            String predictString =
ClassAttribute.value((int)predictedValue) + " (" +
predictedValue + ")";
            result.append("predicted: '" + predictString);

```

```

// check if word is a model word
for (int wordIx = 0; wordIx < splittedText.length;
wordIx++) {
    String sWord = splittedText[wordIx];
    if (modelWords.contains((String)sWord)) {
        gotModelWords++;
        acceptedWordsThisLine.append(sWord + " ");
    }
}
testsWithModelWords[i] =
acceptedWordsThisLine.toString();
}

// should we do something if there is no
modelWords?
if (gotModelWords == 0) {
    result.append("\nWarning!\n\nThe text to classify didn't
contain a single\nword from the modelled words. This makes
it hard for the classifier to\ndo something usefull.\n\nThe result
may be weird.\n\n");
}

try {

    // add the ? class for all test cases
    String[] tmpClassValues = new
String[testText.length];
    for (int i = 0; i < tmpClassValues.length; i++) {
        tmpClassValues[i] = "?";
    }

    for (int i = 0; i < testsWithModelWords.length; i++) {
        Instance inst = new Instance(2);
        inst.setValue(TextAttribute,testsWithModelWords[i]);
        if (tmpClassValues != null && tmpClassValues.length
> 0) {
            inst.setValue(ClassAttribute, tmpClassValues[i]);
        }
        testCases.add(inst);
    }

StringToWordVector filter = null;

// default values according to Java Doc:
int wordsToKeep = 1000;

Instances filtered = null;

try {

    filter = new StringToWordVector(wordsToKeep);
    filter.setOutputWordCounts(true);
    filter.setSelectedRange("1");

    filter.setInputFormat(testCases);

```



```

        filtered = weka.filters.Filter.useFilter(testCases,filter);

    } catch (Exception e) {
        e.printStackTrace();
    }
    Instances filteredTests = filtered;

    int startIx = instances.numInstances();
    String testType = "new case";

    try {

        result.append("\nCHECKING ALL THE
INSTANCES:\n");

        Enumeration enumClasses =
ClassAttribute.enumerateValues();
        result.append("Class values (in order): ");
        while (enumClasses.hasMoreElements()) {
            String classStr =
(String)enumClasses.nextElement();
            result.append("'" + classStr + "' ");
        }
        result.append("\n");

        // startIx is a fix for handling text cases

        for (int i = startIx; i < filteredTests.numInstances();
i++) {

            SparseInstance sparseInst = new
SparseInstance(filteredTests.instance(i));
            sparseInst.setDataset(filteredTests);

            result.append("\nTesting: '" + testText[i-startIx] +
"\n");

            double correctValue =
(double)sparseInst.classValue();
            double predictedValue =
classifier.classifyInstance(sparseInst);

            String predictString =
ClassAttribute.value((int)predictedValue) + " (" +
predictedValue + ")";
            result.append("predicted: '" + predictString);

            // print comparison if not new case
            if (!"newcase".equals(testType)) {
                String correctString =
ClassAttribute.value((int)correctValue) + " (" + correctValue
+ ")";
                String testString = ((predictedValue ==
correctValue) ? "OK!" : "NOT OK!") + "!";
                result.append("'" + real class: '" + correctString + "'
==> " + testString);
            }
            result.append("\n");

```

```

        result.append("\n");
    }
}
catch (Exception e) {
    e.printStackTrace();
    result.append("\nException (sorry!):\n" + e.toString());
}
result.append("\n");

} catch (Exception e) {
    e.printStackTrace();
    result.append("\nException (sorry!):\n" + e.toString());
}
return result.toString();
}
}

```

Listing 3

VI. VERIFICATION OF OUTPUT

The output generated by the three sample codes of web services above is found to be same as that of the java data mining code for the desktop application. This shows that the datasets provided as inputs are correctly mined by the above given sample web services with the same precision and accuracy.

The input, output on testing these three web services with the SOAP request and response messages, the WSDL document and the code for the client is given in the sections below.

J48 Classifier as Web Service

The training data set used here is the IRIS DATASET and the location of the file iris.arff is given as input to the code along with the string parameter specifying the name of the algorithm.

Output of the web service with its corresponding SOAP request and SOAP response is as shown below:

A. execute Method invocation

a) Method parameter(s)

Type	Value
	CLASSIFIER weka.classifiers.trees.J48 -U FILTER
java.lang.String	weka.filters.unsupervised.instance.Randomize DATASET
	C:/Users/s1/Documents/NetBeansProjects/JavaApp/build/classes/iris.arff

b) Method returned

```

java.lang.String : "Weka - Demo ===== Classifier...:
weka.classifiers.trees.J48 -U -M 2 Filter.....:
weka.filters.unsupervised.instance.Randomize -S 42 Training
file:
C:/Users/s1/Documents/NetBeansProjects/JavaApp/build/clas
ses/iris.arff J48 unpruned tree ----- petalwidth <=
0.6: Iris-setosa (50.0) petalwidth > 0.6 | petalwidth <= 1.7 | |
petallength <= 4.9: Iris-versicolor (48.0/1.0) | | petallength >
4.9 | | | petalwidth <= 1.5: Iris-virginica (3.0) | | | petalwidth >
1.5: Iris-versicolor (3.0/1.0) | petalwidth > 1.7: Iris-virginica
(46.0/1.0) Number of Leaves : 5 Size of the tree : 9 Correctly
Classified Instances 142 94.6667 % Incorrectly Classified
Instances 8 5.3333 % Kappa statistic 0.92 Mean absolute error
0.043 Root mean squared error 0.1854 Relative absolute error
9.6778 % Root relative squared error 39.3217 % Total
Number of Instances 150 === Confusion Matrix === a b c <--
classified as 49 1 0 | a = Iris-setosa 0 46 4 | b = Iris-versicolor
0 3 47 | c = Iris-virginica === Detailed Accuracy By Class
=== TP Rate FP Rate Precision Recall F-Measure Class 0.98
0 1 0.98 0.99 Iris-setosa 0.92 0.04 0.92 0.92 0.92 Iris-
versicolor 0.94 0.04 0.922 0.94 0.931 Iris-virginica "

```

c) SOAP Request

```

<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header/>
  <S:Body>
    <ns2:execute xmlns:ns2="http://demo1/">
      <input>CLASSIFIER weka.classifiers.trees.J48 -U
FILTER weka.filters.unsupervised.instance.Randomize
DATASET
C:/Users/s1/Documents/NetBeansProjects/JavaApp/build/clas
ses/iris.arff</input>
    </ns2:execute>
  </S:Body>
</S:Envelope>

```

d) SOAP Response

```

<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:executeResponse xmlns:ns2="http://demo1/">
      <return>Weka - Demo
=====

```

```

Classifier...: weka.classifiers.trees.J48 -U -M 2
Filter.....: weka.filters.unsupervised.instance.Randomize -S
42
Training file:
C:/Users/s1/Documents/NetBeansProjects/JavaApp/build/clas
ses/iris.arff

```

J48 unpruned tree

```

-----
petalwidth <= 0.6: Iris-setosa (50.0)
petalwidth > 0.6
| petalwidth <= 1.7
| | petallength <= 4.9: Iris-versicolor (48.0/1.0)
| | petallength > 4.9
| | | petalwidth <= 1.5: Iris-virginica (3.0)
| | | petalwidth > 1.5: Iris-versicolor (3.0/1.0)
| petalwidth > 1.7: Iris-virginica (46.0/1.0)

```

Number of Leaves : 5

Size of the tree : 9

Correctly Classified Instances	142	94.6667 %
Incorrectly Classified Instances	8	5.3333 %
Kappa statistic	0.92	
Mean absolute error	0.043	
Root mean squared error	0.1854	
Relative absolute error	9.6778 %	
Root relative squared error	39.3217 %	
Total Number of Instances	150	

=== Confusion Matrix ===

```

a b c <-- classified as
49 1 0 | a = Iris-setosa
0 46 4 | b = Iris-versicolor
0 3 47 | c = Iris-virginica

```

=== Detailed Accuracy By Class ===

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.98	0	1	0.98	0.99	Iris-setosa
0.92	0.04	0.92	0.92	0.92	Iris-versicolor
0.94	0.04	0.922	0.94	0.931	Iris-virginica

```

</return>
</ns2:executeResponse>
</S:Body>
</S:Envelope>

```

WSDL:

```

<?xml version="1.0" encoding="UTF-8" ?>
<!--
Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's
version is JAX-WS RI 2.1.3.1-hudson-417-SNAPSHOT.
-->
<!--
Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's
version is JAX-WS RI 2.1.3.1-hudson-417-SNAPSHOT.

```

```

-->
<definitions xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://demo1/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://demo1/"
name="NewWebService12Service">
<types>
<xsd:schema>
<xsd:import namespace="http://demo1/"
schemaLocation="http://localhost:13699/WebApplication12
/NewWebService12Service?xsd=1" />
</xsd:schema>
</types>
<message name="execute">
<part name="parameters" element="tns:execute" />
</message>
<message name="executeResponse">
<part name="parameters"
element="tns:executeResponse" />
</message>
<message name="Exception">
<part name="fault" element="tns:Exception" />
</message>
<portType name="NewWebService12">
<operation name="execute">
<input message="tns:execute" />
<output message="tns:executeResponse" />
<fault message="tns:Exception" name="Exception" />
</operation>
</portType>
<binding name="NewWebService12PortBinding"
type="tns:NewWebService12">
<soap:binding
transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
<operation name="execute">
<soap:operation soapAction="" />
<input>
<soap:body use="literal" />
</input>
<output>
<soap:body use="literal" />
</output>
<fault name="Exception">
<soap:fault name="Exception" use="literal" />
</fault>
</operation>
</binding>
<service name="NewWebService12Service">
<port name="NewWebService12Port"
binding="tns:NewWebService12PortBinding">

```

```

<soap:address
location="http://localhost:13699/WebApplication12/NewW
ebService12Service" />
</port>
</service>
</definitions>

```

Client code:

```

package org.weka;

import demo1.NewWebService12Service;
import java.io.*;
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.xml.ws.WebServiceRef;

public class NewServlet extends HttpServlet {
    @WebServiceRef(wsdlLocation =
"http://localhost:13699/WebApplication12/NewWebService1
2Service?wsdl")
    private NewWebService12Service service;

    /**
     * Processes requests for both HTTP GET
and POST methods.
     * @param request servlet request
     * @param response servlet response
     */
    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {

            // TODO output your page here

            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet NewServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet NewServlet at " +
request.getContextPath () + "</h1>");

        }

        // Call Web Service Operation
        demo1.NewWebService12 port =
service.getNewWebService12Port();
        // TODO initialize WS operation arguments here

        java.lang.String input = "CLASSIFIER
weka.classifiers.trees.J48 -U FILTER

```

```
weka.filters.unsupervised.instance.Randomize DATASET
C:/Users/s1/Documents/NetBeansProjects/JavaApp/build/classes/iris.arff";
```

```
// TODO process result here
```

```
java.lang.String result = port.execute(input);
out.println("Result = "+result);
} catch (Exception ex) {
```

```
// TODO handle custom exceptions here
```

```
}
    out.println("</body>");
    out.println("</html>");
```

```
    } finally {
        out.close();
    }
}
```

```
// <editor-fold defaultstate="collapsed" desc="HttpServlet
methods. Click on the + sign on the left to edit the code.">
```

```
/**
 * Handles the HTTP <code>GET</code> method.
 * @param request servlet request
 * @param response servlet response
 */
```

```
protected void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
    processRequest(request, response);
}
```

```
/**
 * Handles the HTTP <code>POST</code> method.
 * @param request servlet request
 * @param response servlet response
 */
```

```
protected void doPost(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
    processRequest(request, response);
}
```

```
/**
 * Returns a short description of the servlet.
 */
```

```
public String getServletInfo() {
    return "Short description";
}
```

```
// </editor-fold>
}
```

Output for client code:

VII. SERVLET NEWSERVLET AT /WEBAPPLICATIONCLIENT12

```
Result = Weka - Demo ===== Classifier...:
weka.classifiers.trees.J48 -U -M 2 Filter.....:
weka.filters.unsupervised.instance.Randomize -S 42 Training
file:
```

```
C:/Users/s1/Documents/NetBeansProjects/JavaApp/build/classes/iris.arff J48 unpruned tree ----- petalwidth <=
0.6: Iris-setosa (50.0) petalwidth > 0.6 | petalwidth <= 1.7 | |
petallength <= 4.9: Iris-versicolor (48.0/1.0) | | petallength >
4.9 | | | petalwidth <= 1.5: Iris-virginica (3.0) | | | petalwidth >
1.5: Iris-versicolor (3.0/1.0) | petalwidth > 1.7: Iris-virginica
(46.0/1.0) Number of Leaves : 5 Size of the tree : 9 Correctly
Classified Instances 142 94.6667 % Incorrectly Classified
Instances 8 5.3333 % Kappa statistic 0.92 Mean absolute error
0.043 Root mean squared error 0.1854 Relative absolute error
9.6778 % Root relative squared error 39.3217 % Total
Number of Instances 150 === Confusion Matrix === a b c <--
classified as 49 1 0 | a = Iris-setosa 0 46 4 | b = Iris-versicolor
0 3 47 | c = Iris-virginica === Detailed Accuracy By Class
=== TP Rate FP Rate Precision Recall F-Measure Class 0.98
0 1 0.98 0.99 Iris-setosa 0.92 0.04 0.92 0.92 0.92 Iris-
versicolor 0.94 0.04 0.922 0.94 0.931 Iris-virginica
```

EM Clusterer as Web Service

The training data set used here is the WEATHER DATASET and the location of the file weather.arff is given as input to the code.

Output of the web service and its corresponding SOAP request and SOAP response is given below:

A. execute Method invocation

a) Method parameter(s)

Type	Value
java.lang.String	C:/Users/s1/Documents/NetBeansProjects/Clustering Demo/build/classes/weather.arff

b) Method returned

```
java.lang.String : "Weka - Demo ===== --> normal
EM == Number of clusters selected by cross validation: 1
Cluster: 0 Prior probability: 1 Attribute: outlook Discrete
Estimator. Counts = 6 5 6 (Total = 17) Attribute: temperature
Normal Distribution. Mean = 73.5714 StdDev = 6.3326
Attribute: humidity Normal Distribution. Mean = 81.6429
StdDev = 9.9111 Attribute: windy Discrete Estimator. Counts
= 7 9 (Total = 16) Attribute: play Discrete Estimator. Counts
= 10 6 (Total = 16) === Clustering stats for training data ===
Clustered Instances 0 14 (100%) Log likelihood: -9.4063 -->
manual # of clusters: 1 --> density (CV) # of clusters: 0 "
```

c) SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header/>
  <S:Body>
    <ns2:execute xmlns:ns2="http://demo/">

<input>C:/Users/s1/Documents/NetBeansProjects/Clustering
Demo/build/classes/weather.arff</input>
    </ns2:execute>
  </S:Body>
</S:Envelope>
```

--> density (CV)

of clusters: 0

```
</return>
</ns2:executeResponse>
</S:Body>
</S:Envelope>
```

WSDL:

d) SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:executeResponse xmlns:ns2="http://demo/">
      <return>Weka - Demo
```

=====

--> normal

EM

==

Number of clusters selected by cross validation: 1

Cluster: 0 Prior probability: 1

Attribute: outlook

Discrete Estimator. Counts = 6 5 6 (Total = 17)

Attribute: temperature

Normal Distribution. Mean = 73.5714 StdDev = 6.3326

Attribute: humidity

Normal Distribution. Mean = 81.6429 StdDev = 9.9111

Attribute: windy

Discrete Estimator. Counts = 7 9 (Total = 16)

Attribute: play

Discrete Estimator. Counts = 10 6 (Total = 16)

=== Clustering stats for training data ===

Clustered Instances

0 14 (100%)

Log likelihood: -9.4063

--> manual

of clusters: 1

<?xml version="1.0" encoding="UTF-8" ?>

- <!--

Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.3.1-hudson-417-SNAPSHOT.

-->

- <!--

Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.1.3.1-hudson-417-SNAPSHOT.

-->

```
<definitions xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://demo/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://demo/"
name="NewWebService2Service">
```

<types>

<xsd:schema>

<xsd:import namespace="http://demo/"

schemaLocation="http://localhost:13699/WebApplication4/NewWebService2Service?xsd=1" />

</xsd:schema>

</types>

<message name="execute">

<part name="parameters" element="tns:execute" />

</message>

<message name="executeResponse">

<part name="parameters" element="tns:executeResponse" />

</message>

<message name="Exception">

<part name="fault" element="tns:Exception" />

</message>

<portType name="NewWebService2">

<operation name="execute">

<input message="tns:execute" />

<output message="tns:executeResponse" />

<fault message="tns:Exception" name="Exception" />

</operation>

</portType>

<binding name="NewWebService2PortBinding"

type="tns:NewWebService2">

```

<soap:binding
transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
<operation name="execute">
  <soap:operation soapAction="" />
<input>
  <soap:body use="literal" />
</input>
<output>
  <soap:body use="literal" />
</output>
<fault name="Exception">
  <soap:fault name="Exception" use="literal" />
</fault>
</operation>
</binding>
<service name="NewWebService2Service">
<port name="NewWebService2Port"
binding="tns:NewWebService2PortBinding">
  <soap:address
location="http://localhost:13699/WebApplication4/NewWebS
ervice2Service" />
</port>
</service>
</definitions>

```

Client Code:

```

package org.web;

import demo.NewWebService2Service;
import java.io.*;
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.xml.ws.WebServiceRef;

public class NewServlet extends HttpServlet {
    @WebServiceRef(wsdlLocation =
"http://localhost:13699/WebApplication4/NewWebService2S
ervice?wsdl")
    private NewWebService2Service service;

    /**
     * Processes requests for both HTTP <code>GET</code>
and <code>POST</code> methods.
     * @param request servlet request
     * @param response servlet response
     */
    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            //TODO output your page here
            out.println("<html>");

```

```

        out.println("<head>");
        out.println("<title>Servlet NewServlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet    NewServlet    at    "    +
request.getContextPath () + "</h1>");

        try {

            // Call Web Service Operation

            demo.NewWebService2                port                =
service.getNewWebService2Port();

            // TODO initialize WS operation arguments here

            java.lang.String    input    =    "C:/Users/s1/
Documents/NetBeansProjects/ClusteringDemo/build/classes/
weather.arff";

            // TODO process result here

            java.lang.String result = port.execute(input);
            out.println("Result = "+result);
        } catch (Exception ex) {

            // TODO handle custom exceptions here

        }

        out.println("</body>");
        out.println("</html>");

        } finally {
            out.close();
        }
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet
methods. Click on the + sign on the left to edit the code.">
    /**
     * Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Handles the HTTP <code>POST</code> method.
     * @param request servlet request
     * @param response servlet response
     */

```

```

protected void doPost(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    processRequest(request, response);
}

/**
 * Returns a short description of the servlet.
 */

public String getServletInfo() {
    return "Short description";
}
// </editor-fold>
}

```

Output for client code:

VIII.SERVLET NEWSERVLET AT /WEBAPPLICATION5

Result = Weka - Demo ===== --> normal EM ==
 Number of clusters selected by cross validation: 1 Cluster: 0
 Prior probability: 1 Attribute: outlook Discrete Estimator.
 Counts = 6 5 6 (Total = 17) Attribute: temperature Normal
 Distribution. Mean = 73.5714 StdDev = 6.3326 Attribute:
 humidity Normal Distribution. Mean = 81.6429 StdDev =
 9.9111 Attribute: windy Discrete Estimator. Counts = 7 9
 (Total = 16) Attribute: play Discrete Estimator. Counts = 10 6
 (Total = 16) === Clustering stats for training data ===
 Clustered Instances 0 14 (100%) Log likelihood: -9.4063 -->
 manual # of clusters: 1 --> density (CV) # of clusters: 0

Text Classifier as Web Service

The testing and training data set used here are provided in the code itself.

Output of the above web service with Lazy-IBk algorithm and its corresponding SOAP request and SOAP response is given below:

A. execute Method invocation

a) Method parameter(s)

Type	Value
java.lang.String	

b) Method returned

java.lang.String : "dataset: DATA SET: @relation 'data set'
 @attribute text string @attribute class {'no spam','?',spam}
 @data 'hey, buy this from me!',spam 'do you want to
 buy?',spam 'I have a party tonight!','no spam' 'today it is a nice
 weather','no spam' 'you are best',spam 'I have a horse','no
 spam' 'you are my friend','no spam' 'buy, buy, buy!',spam 'it is
 spring in the air','no spam' 'do you want to come?','no spam'
 INFORMATION ABOUT THE CLASSIFIER AND

EVALUATION: classifier.toString(): IB1 instance-based
 classifier using 1 nearest neighbour(s) for classification
 evaluation.toSummaryString(title, false): Summary Correctly
 Classified Instances 10 100 % Incorrectly Classified Instances
 0 0 % Kappa statistic 1 Mean absolute error 0.1026 Root
 mean squared error 0.1088 Relative absolute error 29.4118 %
 Root relative squared error 26.9191 % Total Number of
 Instances 10 evaluation.toMatrixString(): === Confusion
 Matrix === a b c <-- classified as 6 0 0 | a = no spam 0 0 0 | b
 = ? 0 0 4 | c = spam evaluation.toClassDetailsString(): Details
 TP Rate FP Rate Precision Recall F-Measure Class 1 0 1 1 1
 no spam 0 0 0 0 0 ? 1 0 1 1 1 spam
 evaluation.toCumulativeMarginDistribution: -1 0 0.768 100
 CHECKING ALL THE INSTANCES: Class values (in
 order): 'no spam' '?' 'spam' Testing: 'hey, buy this from me!'
 predicted: 'spam (2.0)' real class: 'spam (2.0)' ==> OK!!
 Testing: 'do you want to buy?' predicted: 'spam (2.0)' real
 class: 'spam (2.0)' ==> OK!! Testing: 'I have a party tonight!'
 predicted: 'no spam (0.0)' real class: 'no spam (0.0)' ==> OK!!
 Testing: 'today it is a nice weather' predicted: 'no spam (0.0)'
 real class: 'no spam (0.0)' ==> OK!! Testing: 'you are best'
 predicted: 'spam (2.0)' real class: 'spam (2.0)' ==> OK!!
 Testing: 'I have a horse' predicted: 'no spam (0.0)' real class:
 'no spam (0.0)' ==> OK!! Testing: 'you are my friend'
 predicted: 'no spam (0.0)' real class: 'no spam (0.0)' ==> OK!!
 Testing: 'buy, buy, buy!' predicted: 'spam (2.0)' real class:
 'spam (2.0)' ==> OK!! Testing: 'it is spring in the air'
 predicted: 'no spam (0.0)' real class: 'no spam (0.0)' ==> OK!!
 Testing: 'do you want to come?' predicted: 'no spam (0.0)' real
 class: 'no spam (0.0)' ==> OK!! NEW CASES CHECKING
 ALL THE INSTANCES: Class values (in order): 'no spam' '?'
 'spam' Testing: 'you want to buy from me?' predicted: 'spam
 (2.0)' real class: '?' (1.0)' ==> NOT OK!! Testing: 'usually I
 run in stairs' predicted: 'spam (2.0)' real class: '?' (1.0)' ==>
 NOT OK!! Testing: 'buy it now!' predicted: 'spam (2.0)' real
 class: '?' (1.0)' ==> NOT OK!! Testing: 'buy, buy, buy!'
 predicted: 'spam (2.0)' real class: '?' (1.0)' ==> NOT OK!!
 Testing: 'you are the best, buy!' predicted: 'spam (2.0)' real
 class: '?' (1.0)' ==> NOT OK!! Testing: 'it is spring in the air'
 predicted: 'no spam (0.0)' real class: '?' (1.0)' ==> NOT OK!! "

c) SOAP Request

```

<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header/>
  <S:Body>
    <ns2:execute xmlns:ns2="http://demo/">
  </S:Body>
</S:Envelope>

```

d) SOAP Response

```

<?xml version="1.0" encoding="UTF-8"?>

```

```
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:executeResponse xmlns:ns2="http://demo/">
      <return>dataset:
```

DATA SET:

@relation 'data set'

@attribute text string

@attribute class {'no spam','?',spam}

@data

```
'hey, buy this from me!',spam
'do you want to buy?',spam
'I have a party tonight!','no spam'
'today it is a nice weather','no spam'
'you are best',spam
'I have a horse','no spam'
'you are my friend','no spam'
'buy, buy, buy!',spam
'it is spring in the air','no spam'
'do you want to come?','no spam'
```

INFORMATION ABOUT THE CLASSIFIER AND EVALUATION:

classifier.toString():

IB1 instance-based classifier

using 1 nearest neighbour(s) for classification

evaluation.toSummaryString(title, false):

Summary

Correctly Classified Instances	10	100	%
Incorrectly Classified Instances	0	0	%
Kappa statistic	1		
Mean absolute error	0.1026		
Root mean squared error	0.1088		
Relative absolute error	29.4118 %		
Root relative squared error	26.9191 %		
Total Number of Instances	10		

evaluation.toMatrixString():

=== Confusion Matrix ===

a b c <-- classified as

6 0 0 | a = no spam

0 0 0 | b = ?

0 0 4 | c = spam

evaluation.toClassDetailsString():

Details

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
1	0	1	1	1	no spam
0	0	0	0	?	
1	0	1	1	1	spam

evaluation.toCumulativeMarginDistribution:

```
-1    0
0.768 100
```

CHECKING ALL THE INSTANCES:

Class values (in order): 'no spam' '?' 'spam'

Testing: 'hey, buy this from me!'

predicted: 'spam (2.0)' real class: 'spam (2.0)' ==> OK!!

Testing: 'do you want to buy?'

predicted: 'spam (2.0)' real class: 'spam (2.0)' ==> OK!!

Testing: 'I have a party tonight!'

predicted: 'no spam (0.0)' real class: 'no spam (0.0)' ==> OK!!

Testing: 'today it is a nice weather'

predicted: 'no spam (0.0)' real class: 'no spam (0.0)' ==> OK!!

Testing: 'you are best'

predicted: 'spam (2.0)' real class: 'spam (2.0)' ==> OK!!

Testing: 'I have a horse'

predicted: 'no spam (0.0)' real class: 'no spam (0.0)' ==> OK!!

Testing: 'you are my friend'

predicted: 'no spam (0.0)' real class: 'no spam (0.0)' ==> OK!!

Testing: 'buy, buy, buy!'

predicted: 'spam (2.0)' real class: 'spam (2.0)' ==> OK!!

Testing: 'it is spring in the air'

predicted: 'no spam (0.0)' real class: 'no spam (0.0)' ==> OK!!

Testing: 'do you want to come?'

predicted: 'no spam (0.0)' real class: 'no spam (0.0)' ==> OK!!

NEW CASES

CHECKING ALL THE INSTANCES:

Class values (in order): 'no spam' '?' 'spam'

Testing: 'you want to buy from me?'

predicted: 'spam (2.0)' real class: '?' (1.0)' ==> NOT OK!!

Testing: 'usually I run in stairs'

predicted: 'spam (2.0)' real class: '?' (1.0)' ==> NOT OK!!

Testing: 'buy it now!'

predicted: 'spam (2.0)' real class: '?' (1.0)' ==> NOT OK!!

Testing: 'buy, buy, buy!'

predicted: 'spam (2.0)' real class: '?' (1.0)' ==> NOT OK!!

Testing: 'you are the best, buy!'

predicted: 'spam (2.0)' real class: '?' (1.0)' ==> NOT OK!!

Testing: 'it is spring in the air'

predicted: 'no spam (0.0)' real class: '?' (1.0)' ==> NOT OK!!

```
</return>
  </ns2:executeResponse>
</S:Body>
</S:Envelope>
```

WSDL:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <!--
Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's
version is JAX-WS RI 2.1.3.1-hudson-417-SNAPSHOT.
-->
- <!--
Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's
version is JAX-WS RI 2.1.3.1-hudson-417-SNAPSHOT.
-->
<definitions xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://demo/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://demo/"
name="TextClassifierServiceService">
  <types>
  <xsd:schema>
    <xsd:import namespace="http://demo/"
schemaLocation="http://localhost:13699/TextClassifierServ
ice/TextClassifierServiceService?xsd=1" />
    </xsd:schema>
  </types>
  <message name="execute">
    <part name="parameters" element="tns:execute" />
  </message>
  <message name="executeResponse">
    <part name="parameters" element="tns:executeResponse" />
```

```
</message>
<portType name="TextClassifierService">
  <operation name="execute">
    <input message="tns:execute" />
    <output message="tns:executeResponse" />
  </operation>
</portType>
<binding name="TextClassifierServicePortBinding"
type="tns:TextClassifierService">
  <soap:binding
transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
  <operation name="execute">
    <soap:operation soapAction="" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
</binding>
<service name="TextClassifierServiceService">
  <port name="TextClassifierServicePort"
binding="tns:TextClassifierServicePortBinding">
    <soap:address
location="http://localhost:13699/TextClassifierService/Text
ClassifierServiceService" />
  </port>
</service>
</definitions>
```

Client code:

```
package org.web;

import demo.TextClassifierServiceService;
import java.io.*;
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.xml.ws.WebServiceRef;

public class NewServlet extends HttpServlet {
    @WebServiceRef(wsdlLocation =
"http://localhost:13699/TextClassifierService/TextClassifierSe
rviceService?wsdl")
    private TextClassifierServiceService service;

    /**
     * Processes requests for both HTTP <code>GET</code>
and <code>POST</code> methods.
     * @param request servlet request
     * @param response servlet response
```

```

*/

protected void processRequest(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        // TODO output your page here

        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet NewServlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet NewServlet at " +
request.getContextPath () + "</h1>");

    }

    try {

// Call Web Service Operation

        demo.TextClassifierService port
service.getTextClassifierServicePort();

        // TODO initialize WS operation arguments here

        java.lang.String input = "weka.classifiers.lazy.IBk";

        // TODO process result here

        java.lang.String result = port.execute(input);
        out.println("Result = "+result);
    } catch (Exception ex) {

        // TODO handle custom exceptions here
    }

        out.println("</body>");
        out.println("</html>");

    } finally {
        out.close();
    }
}

// <editor-fold defaultstate="collapsed"
desc="HttpServletRequest methods. Click on the + sign on the left
to edit the code.">
/**
 * Handles the HTTP <code>GET</code> method.
 * @param request servlet request
 * @param response servlet response
 */

protected void doGet(HttpServletRequest request,
    HttpServletResponse response)

```

```

    throws ServletException, IOException {
        processRequest(request, response);
    }

/**
 * Handles the HTTP <code>POST</code> method.
 * @param request servlet request
 * @param response servlet response
 */

protected void doPost(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

/**
 * Returns a short description of the servlet.
 */

    public String getServletInfo() {
        return "Short description";
    }
} // </editor-fold>

```

Output for the client code:

IX. SERVLET NEWSERVLET AT /TEXTCLIENT

Result = dataset: DATA SET: @relation 'data set'
@attribute text string @attribute class {'no spam','?',spam}
@data 'hey, buy this from me!',spam 'do you want to
buy?',spam 'I have a party tonight!','no spam' 'today it is a nice
weather','no spam' 'you are best',spam 'I have a horse','no
spam' 'you are my friend','no spam' 'buy, buy, buy!',spam 'it is
spring in the air','no spam' 'do you want to come?','no spam'
INFORMATION ABOUT THE CLASSIFIER AND
EVALUATION: classifier.toString(): IB1 instance-based
classifier using 1 nearest neighbour(s) for classification
evaluation.toSummaryString(title, false): Summary Correctly
Classified Instances 10 100 % Incorrectly Classified Instances
0 0 % Kappa statistic 1 Mean absolute error 0.1026 Root
mean squared error 0.1088 Relative absolute error 29.4118 %
Root relative squared error 26.9191 % Total Number of
Instances 10 evaluation.toMatrixString(): === Confusion
Matrix === a b c <-- classified as 6 0 0 | a = no spam 0 0 0 | b
= ? 0 0 4 | c = spam evaluation.toClassDetailsString(): Details
TP Rate FP Rate Precision Recall F-Measure Class 1 0 1 1 1
no spam 0 0 0 0 0 0 ? 1 0 1 1 1 spam
evaluation.toCumulativeMarginDistribution: -1 0 0.768 100
CHECKING ALL THE INSTANCES: Class values (in
order): 'no spam' '?' 'spam' Testing: 'hey, buy this from me!'
predicted: 'spam (2.0)' real class: 'spam (2.0)' ==> OK!!
Testing: 'do you want to buy?' predicted: 'spam (2.0)' real
class: 'spam (2.0)' ==> OK!! Testing: 'I have a party tonight!'
predicted: 'no spam (0.0)' real class: 'no spam (0.0)' ==> OK!!
Testing: 'today it is a nice weather' predicted: 'no spam (0.0)'

real class: 'no spam (0.0)' ==> OK!! Testing: 'you are best' predicted: 'spam (2.0)' real class: 'spam (2.0)' ==> OK!! Testing: 'I have a horse' predicted: 'no spam (0.0)' real class: 'no spam (0.0)' ==> OK!! Testing: 'you are my friend' predicted: 'no spam (0.0)' real class: 'no spam (0.0)' ==> OK!! Testing: 'buy, buy, buy!' predicted: 'spam (2.0)' real class: 'spam (2.0)' ==> OK!! Testing: 'it is spring in the air' predicted: 'no spam (0.0)' real class: 'no spam (0.0)' ==> OK!! Testing: 'do you want to come?' predicted: 'no spam (0.0)' real class: 'no spam (0.0)' ==> OK!! NEW CASES CHECKING ALL THE INSTANCES: Class values (in order): 'no spam' '?' 'spam' Testing: 'you want to buy from me?' predicted: 'spam (2.0)' real class: '?' (1.0)' ==> NOT OK!! Testing: 'usually I run in stairs' predicted: 'spam (2.0)' real class: '?' (1.0)' ==> NOT OK!! Testing: 'buy it now!' predicted: 'spam (2.0)' real class: '?' (1.0)' ==> NOT OK!! Testing: 'buy, buy, buy!' predicted: 'spam (2.0)' real class: '?' (1.0)' ==> NOT OK!! Testing: 'you are the best, buy!' predicted: 'spam (2.0)' real class: '?' (1.0)' ==> NOT OK!! Testing: 'it is spring in the air' predicted: 'no spam (0.0)' real class: '?' (1.0)' ==> NOT OK!!

X. CONCLUSION

Weka is a widely used tool for data mining. It provides with an open source code that can be used for machine learning. Not only that the interface provided by the software can also be used for mining large datasets. However keeping in mind the benefits of a web service as compared to a desktop application, we have demonstrated how Weka (or any other software) can be ported as a web service. The results obtained from the web application are the same as that of the desktop application, thereby highlighting the fact that the data can be mined by the Weka web service in a similar fashion as can be mined by Weka desktop application, with easier access to large datasets and lesser memory consumption of the system.

REFERENCES

- [1] Witten Ian H, and Frank Eibe, Data Mining Practical Machine Learning Tools and Techniques, Academic Press, pp 14-395.
- [2] Frank Eibe, Hall, Trigg, Holmes, Data Mining in Bioinformatics using Weka, pp.1-2. Bioinformatics Volume:20, Issue:15, Pages: 2479-2481 ISSN: 1367-4803, ISBN 1460-2059.
- [3] Bill.Palace, *Technology note prepared for Management 274A, 1996.* from <http://www.anderson.ucla.edu/faculty/jason.frand/teacher/technologies/palace/datamining.htm>
- [4] University of Waikato, Weka 3: Data Mining Software in Java. from <http://www.cs.waikato.ac.nz/~ml/weka/http://www.cs.waikato.ac.nz/~ml/weka/>
- [5] Haridas Mandar, CIS764-Step By Step Tutorial for Weka, 2008. <http://www.docstoc.com/docs/2582601/CIS764---Step-By-Step-Tutorial-for-Weka-By-Mandar-Haridas>.
- [6] Markov Zdravko, and Russell Ingrid, An Introduction to the WEKA Data Mining System, 2006, Proceedings of the 11th annual SIGCSE conference on Innovation and technology in Computer Science Education, Bologna, Italy, 367-368.
- [7] <http://en.wikipedia.org/wiki/WebServices/>
- [8] Dimov Rossen, WEKA: Practical Machine Learning Tools and Techniques in Java. from http://www.dfki.de/~kipp/seminar_ws0607/slides/Dimov_WEKA.pdf.
- [9] Statsoft Electronic textbook on cluster analysis from: <http://www.statsoft.com/textbook/cluster-analysis/>
- [10] Witten Ian H, and Frank Eibe, WEKA Machine Learning Algorithms in Java.
- [11] Mark F.Hornick, Eric Marcade, Sunil Venkayala, Java Data Mining Strategy, Standard, and Practice, Morgan Kaufmann Series, pp 3-116.
- [12] http://en.wikipedia.org/wiki/K-nearest_neighbor_algorithm