ATINER CONFERENCE PRESENTATION SERIES No: COM2018-0124

ATINER's Conference Paper Proceedings Series
COM2018-0124
Athens, 16 November 2018

Appropriate System to Support Decision Making in Medicine Area

Ana Ktona

Athens Institute for Education and Research
8 Valaoritou Street, Kolonaki, 10683 Athens, Greece

ATINER's Conference Paper Proceedings Series
COM2018-0124
Athens, 16 November 2018
ISSN: 2529-167X

Ana Ktona, Associate Professor, University of Tirana, Albania

Appropriate System to Support Decision Making in Medicine Area

ABSTRACT

Clinical Decision Support Systems, which support physicians in making diagnostic and therapeutic decisions by providing to them passive and active referential information as well as reminders, alerts, and guidelines, attract much interest. Theoretical and practical researches have shown that well designed Clinical decision support systems help physician in disease diagnosis, treatment options, etc. There are two types of Clinical Decision Support Systems in medicine area: Knowledge-based CDSS and Non-knowledge-based CDSS. The logic of Knowledge-based CDSS is based on rules and associations of data in the form of IF … THEN. These rules could be found by algorithms. The logic of Non-knowledge-based CDSS is based on models created by Machine Learning algorithms. Both types of CDSS have advantages and disadvantages. But, how do they compare when evaluated against each other? This paper will identify their logic performance through experimentation and analysis in order to compare them objectively. A representative algorithm will be chosen for each of these Clinical decision support systems, and then it will be run against three patient diagnosis datasets: diabetes diagnosis, contact lenses recommendation and breast cancer diagnosis. Data will be gathered on algorithm accuracy, for a varying dataset size. It is expected that knowledge based CDSS perform better in small dataset and non-knowledge based CDSS perform better in large datasets.

Keywords: Knowledge-based CDSS, Non-knowledge-based CDSS, Machine Learning.

Introduction

In the healthcare system, a large amount of electronic data is produced every day as a result of various examinations and treatments. Raw produced data, which has features that make it more challenging their studying and analyzing, can convert to incredibly useful information. This information helps to make decisions when determining the diagnosis and / or suggestion of treatment, leading to reduction of diagnosis errors and increase of quality in treating diseases.

Electronic systems that allow the discovering of this useful information that helps in clinical decision making (The process of formulating a diagnosis is called clinical decision making)[1] has become a necessity in the medical field. This is because:

- There are a lot of diseases, the countless symptoms and tests, and too many treatment options that make up the medical body of knowledge.
- The complexity of the human physiology.
- Absence of absolutes in medicine.

Computerized Decision Support Systems are developed to address this necessity in medicine. These systems are also called Clinical Decision Support Systems. Some of the more quoted definitions about Clinical Decision Support Systems are:

"Clinical decision support systems link health observations with health knowledge to influence health choices by clinicians for improved health care."[2]

"Clinical Decision Support Systems are active knowledge systems which use two or more items of patient data to generate case-specific advice."[3]

Clinical decision support (CDS) provides clinicians, staff, patients or other individuals with knowledge and person-specific information, intelligently filtered or presented at appropriate times, to enhance health and better health care.[4]

The main purpose of a Clinical Decision Support Systems is to help a physician by suggesting a diagnosis based on findings and/or suggesting a treatment at the point of health care.

There are two types of Clinical Decision Support Systems:

- Knowledge – Based CDSS
- Non Knowledge -Based CDSS

[1]Encyclopedia Britannica: https://www.britannica.com/science/clinical-decision-making.
[2]Robert Hayward, Centre for Health Evidence http://www.cche.net/.
[3]http://www.openclinical.org/dss.html.
[4]https://www.healthit.gov/providers-professionals/faqs/what-clinical-decision-support.

Literature Review

CDSS have been shown to improve physicians' performance significantly (Angela L. P. Chow, 2015; Clemens, 2018; Garg AX, 2005; Nachtigall, 2014).

The aim of the study (Angela L. P. Chow, 2015) was to evaluate the effectiveness of the hospital's antibiotic CDSS on patients' clinical outcomes, and the modification of these effects by patient factors. They conducted a prospective cohort study in a tertiary-care hospital in Singapore. They used multilevel logistic regression model and found that receipt of CDSS-recommended antibiotics reduced mortality risk in patients aged 65 years or younger and did not increase the risk in older patients. They suggested that physicians should be informed of the benefits to increase their acceptance of CDSS recommendations.

The aim of the study (Clemens, 2018) was to analyze the current literature for the impact of HIT on medical outcomes. They analyzed publications that defined an HIT intervention and an effect on medical outcomes in terms of efficiency or effectiveness from Cumulative Index of Nursing and Allied Health Literature (CINAHL) and Medical Literature Analysis and Retrieval System Online (MEDLINE) databases. They found at least one improved medical outcome as a result of HIT adoption in 81% of research studies that met inclusion criteria and no statistical difference in outcomes as a result of HIT in 19% of included studies. They concluded that a strong majority of the literature shows positive effects of HIT on the effectiveness of medical outcomes, which positively supports efforts that prepare for stage 3 of meaningful use.

The aim of the study (Garg AX, 2005) was to review controlled trials assessing the effects of computerized clinical decision support systems (CDSSs) and to identify study characteristics predicting benefit. They used data from MEDLINE, EMBASE, Cochrane Library, Inspec, and ISI databases They compared CDSS with care provided without a CDSS on practitioner performance or patient outcomes and found that the CDSS improved practitioner performance in 64% of the studies assessing this outcome, including 40% of diagnostic systems, 76% of reminder systems, 62% of disease management systems, and 66% of drug-dosing or prescribing systems.

The aim of the study (Nachtigall, 2014) was to evaluate long-term effects of using a CDSS in support of rational anti-infective treatment strategies based on guidelines. They implement a CDSS to evaluate these effects and concluded that: "Implementation of computerized regional adapted guidelines for antibiotic therapy is paralleled with improved adherence. Even without further measures, adherence stayed high for a longer period and was paralleled by reduced antibiotic exposure. Improved guideline adherence was associated with reduced ICU mortality."

Knowledge-Based CDSS

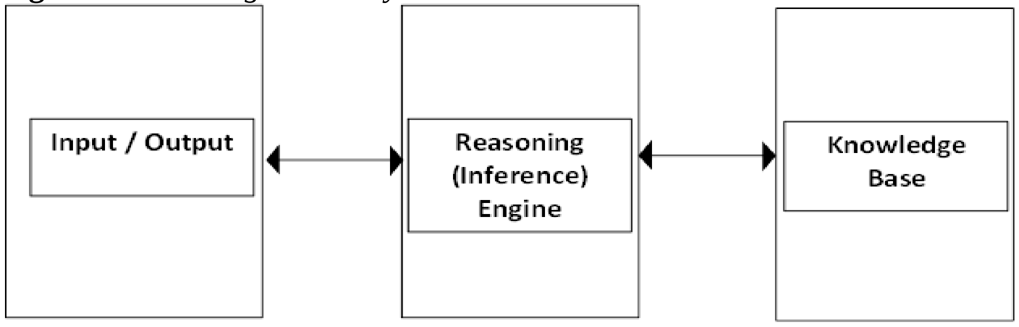
The majority of knowledge-based systems are derived from early expert systems. The expert systems aimed at simulating human thinking in data processing. The characteristics of these systems, which tried to provide the logic and reasoning of a human decision maker, are: heuristic, transparent and flexible

- A heuristic system reasons with judgmental knowledge and with formal knowledge of established theories;
- A transparent system provides explanations of its line of reasoning and answers to queries about its knowledge;
- A flexible system integrates new knowledge incrementally into its existing store of knowledge.

A lot of Knowledge-Based CDSS are implemented in the medicine field due to the intuitive perception that these systems improve healthcare quality. It is widely believed that medicine profit more than the other fields from these systems.

It is assumed that a computer could imitate the thought processes of a real-life physician. The computer then could give a range of diagnosis or treatments based on the information at hand. The physician evaluates the symptoms of a patient, personal testimonies and uses the systems as a reference for possible diagnosis or treatments.

Figure 1. *Knowledge Based Systems Architecture*



Knowledge-based CDSS use traditional Artificial Intelligence (Russell, 2016) methods to suggest diagnosis or treatment. Conditional logic is the most used traditional Artificial Intelligence method in Knowledge-based CDSS. As it is seen from the Figure 1 there are three main parts in a knowledge-based CDSS: the user interface (Input/Output), the inference (reasoning) engine and the knowledge base.

The knowledge base contains information (general principles) which is represented in the form of IF-THEN rules. The statement, or set of statements, after the word if represents some pattern which is observed and the statement, or set of statements, after the word then represents some conclusion that can be drawn or some action that should be taken. A knowledge base could work in conjunction with an algorithmic structure for data analysis.

Part of the knowledge base CDSS is also a collection of specific details that apply to the current patient (sometimes called also working memory).

Inference engine processes information from the knowledge base. It integrates the rules in the knowledge base with patient's current data, allowing the system to create suitable rules and conditions for the patient based on his/her medical history and the severity of patient's current condition. An important aspect of the KBS architecture is that the inference engine and knowledge base are separated. There are a lot of benefits from this architecture. Some of them are:

- The reasoning mechanism could be stable.
- The knowledge base is able to grow and change as knowledge is added.

User interface (input/ output) is the "place" where the physician inputs the data of a patient and receives the corresponding results. The patient data can be entered manually or provided through a computer-based record. Generally the results (output) are represented in the form of recommendations or alerts allowing the physician to take the final decision about the diagnosis or treatment.

KB CDSS advantages include documentation of knowledge, self-learning, explanation, intelligent support in decision making and reasoning. On the other hand the development of KB CDSS is very time consuming.

Non-Knowledge-Based CDSS

Non knowledge-based CDSS are systems that aim to learn from past experiences. They use some machine learning algorithms (Tanveer, 2015) in historical clinical data to find useful patterns in them. These learned lessons are saved in knowledge base and could be used in actual patient data. In contrast with knowledge based systems these systems do not provide explicit information on how their conclusions are drawn. They don't explain the reason behind them. Non-knowledge-based systems focus mainly on a single disease.

The most popular types of Non knowledge-based CDSSs are:

- CDSS that use Artificial Neural Networks learning algorithm.
- CDSS that use as learning algorithms Genetic Algorithms.

Artificial Neural Networks

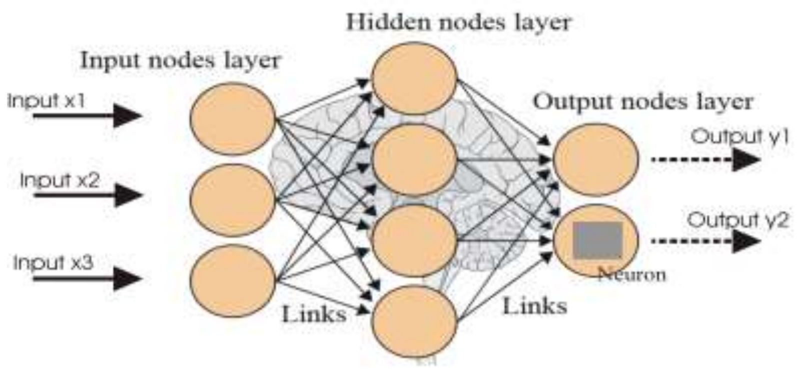
Neural networks represent a metaphor of brain for information processing. They are seen as a system, backed by the results of human brain research, with a structure similar to that of biological nerve networks, Artificial Neural Networks (ANN) simulate human thinking by evaluating and eventually learning from existing examples/occurrences (Yegnanarayana, 2009). Neural networks, though have a biologically inspired modeling skills, are essentially statistical modeling tools. These models are not actually an exact copy of the brain's functioning.

Each Artificial neural network is composed of a collection of neurons, grouped in layers. Three layers are known: input, intermediate (called hidden layer) and output. A hidden layer is a layer of neurons that gets data from the previous layer and converts inputs to the output for further processing. Some hidden layers can be placed between the input layer and the output layer, although the use of only one hidden layer is most often used. In this case, the hidden layer simply converts the inputs to a nonlinear combination and passes the transformed inputs to the output layer. The most common hidden layer interpretation is a feature extraction mechanism. That is, the hidden layer converts the original problem inputs into some of the highest-level combinations of such inputs.

Like a neural biological network, an ANN can be organized in different ways (e.g., topology or architecture); which means that neurons can be linked in different ways. Therefore, ANN appears in many configurations called architectures. When information is processed, many of the processing elements perform their calculations at the same time. This parallel processing resembles the way the brain works and this differs from serial processing of conventional informatics.

An ANN finds patterns in the patient data and then correlations between the signs/symptoms of a patient and a possible diagnosis. They normally are developed for one disease. Like every system that finds patterns in historical data a large amount of clinical data has to be inputted in the neural network. These data are used to “train” the network: i.e. it analyses them and then hypothesizing the correct output (This is the set of weights for the links between nodes.)

Figure 2. *Structure of Artificial Neural Network*



Source: exploreai.[5]

Learned results are compared to actual results (the difference between actual and learned weights is made). The network adjust the results following this process until a substantial number of correct predictions is made

The advantage of using ANN is that it can analyze incomplete data by reasoning what the data should be. With the increase of patient data being analyzed the quality of processes and analysis by ANN is improved. On the other hand the weights produced by these systems are not easily interpreted and the process of “training” is time consuming.

Genetic Algorithms

The basic idea of a genetic algorithm is to apply biological principles of natural evolution in artificial systems. A genetic algorithm is an iterative procedure

[5]https://exploreai.org/p/machine-learning-introduction-to-machine-learning.

involving a population of individuals, each of which is represented by a finite symbol string, known as genes, encoding a possible solution in the space of a given problem. This space, referred to as the search space, summarizes all possible solutions to the problem. Genetic algorithms are mainly applied to areas that are too big to be sought in a finite way.

The Genetic Algorithm is a method based on examples of natural selection in the "real natural world" (Goldberg, 1989). GA uses different methodologies to create more and more consistent solutions to a given problem by using older (parent) solutions to create new ones (child). This will lead to an approximate solution of the given solution space. It is believed that like in the nature over time this will lead to solutions that have more and more skills to adapt, and better survival opportunities.

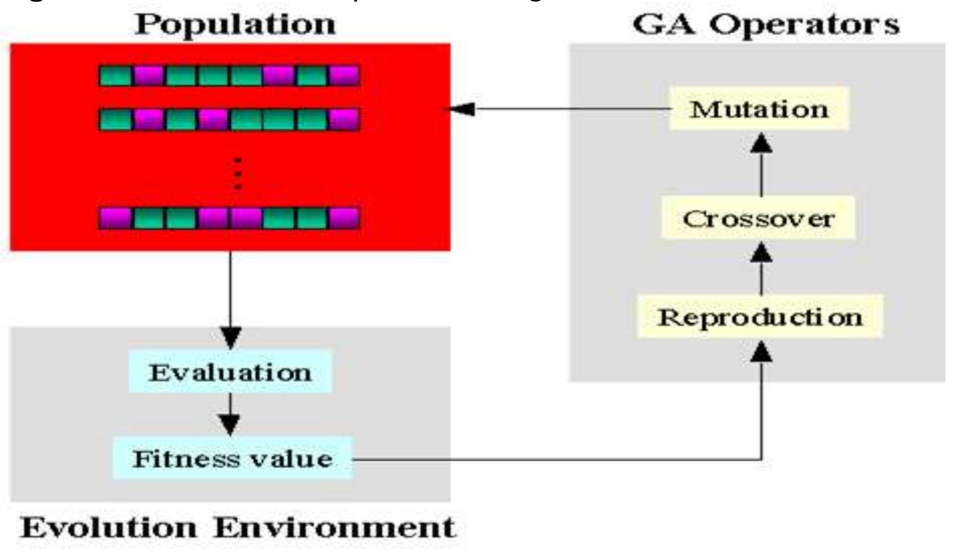
There are many different implementations of Genetic Algorithms. Because of the complexity in deriving good parameters, researchers prefer to use a more traditional, algorithm instead of implementing their algorithm for any new problem.

Even the simplest Genetic Algorithm for a given problem needs to have a set of five components to work properly (Chapman, 2001).

- A genetic representation of potential problem solutions.
- One way to create an initial population for potential solutions.
- An assessment function that plays the role of the environment, ranking the solutions in terms of their fitness with the environment.
- Genetic operators that change individuals to create children.
- Values for the different parameters that the Genetic Algorithm uses (population size, probabilities of application of genetic operators, etc.).

Genetic algorithms attempt to solve a problem by using randomly generated solutions to that problem. The random sets of solutions to the problem are evaluated for their quality through a “fitness function”. The more fit solutions are in the first places and new solutions could be created by combining them. New solutions are evaluated in the same way as their parent. The process is repeated over and over until an optimal solution is found. Genetic Algorithms in medicine attempt to achieve optimal diagnostic and treatment results by following the described process.

Figure 3. *Evolution Flow of a Genetic Algorithm*



Source: (Ying-Hong, 2001).

Genetic algorithms and neural networks are functionally similar in that they are "black boxes" that attempt to derive knowledge from patient data.

Methodology

The logic of Knowledge-based CDSS is based on rules, that can be found by algorithms and associations of data in the form of IF … THEN. On the other hand, the logic of Non-knowledge-based CDSS is based on models created by Machine Learning algorithms. In order to compare them objectively, and as a result to give an answer to the research question, i.e. what is their performance when evaluated against each other in a controlled environment, is identified their logic performance through experimentation and analysis.

Experiments are carried out through Waikato Environment for Knowledge Analysis (WEKA)[6] that has as a programming language Java and its variants based on GNU (General Public License). There are three main ways of using WEKA.

- First is the analysis of the results of the methods to learn more about the data, to get as much information about them.
- Second, it is the creation of a model for predicting new cases
- Third, comparison of methods.

Experiments have been carried out against three patient diagnosis datasets: diabetes diagnosis, contact lenses recommendation and breast cancer diagnosis.

[6]http://www.cs.waikato.ac.nz/ml/weka/index.html.

Knowledge-Based CDSS vs Non-Knowledge-Based CDSS

A lot of algorithms can be used to represent the knowledge through rules. We can mention Decision Table, OneR, JRip, M5Rules PART etc. To represent Knowledge Based CDSS was chosen PART algorithm because it had the best performance when was compared to other candidate algorithms to represent Knowledge Based CDSS. As mentioned before, two well-known algorithms that can be used to represent Non-Knowledge Based CDSS are Artificial Neural Networks and Genetic Algorithms but the last ones have seen less. As a result Artificial Neural Networks were used to represent Non-Knowledge Based CDSS.

PART Algorithm

PART algorithm generates an unrestricted decision list by building a partial decision tree to obtain each rule.[7] PART uses ID3 algorithm to build a partial decision tree which have nodes and leaves nodes are used for the tests (part between IF … Then) and leaves for the conclusion (part after THEN). ID3 algorithm is the traditional decision trees algorithm (Quinlan, 1979; Quinlan, 1983). One possible implementation of this algorithm could be found in Annex A. The main task of the ID3 algorithm is to determine the feature to be tested on each tree node. Below is the pseudo code for building a partial tree:

```
Procedure Expand Subset (S)

Choose the feature to be tested that split the given set of examples into
subsets
While exists subsets that are not expanded
And all the expanded subsets are leaves
Choose next subset t and expand it
If all the expanded subsets are leaves and
Estimated error for partial tree>= estimated error for node
Undo expansion into subsets and make node a leaf
```

Multilayer Perceptron

One implementation of artificial neural networks is multilayer perceptron (MLP). These structural models consist of multiple layers of neurons. The information goes through the neural network in one direction, from the input layers of the network, through one or more hidden layers, toward the output layer. The neurons of each layer are bound only to neurons of the subsequent layer. Weka has an implementation of Multilayer Perceptron.[8] One possible implementation of this algorithm could be found in Annex B.

[7]http://weka.sourceforge.net/doc.dev/weka/classifiers/rules/PART.html.
[8]http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/MultilayerPerceptron.html.

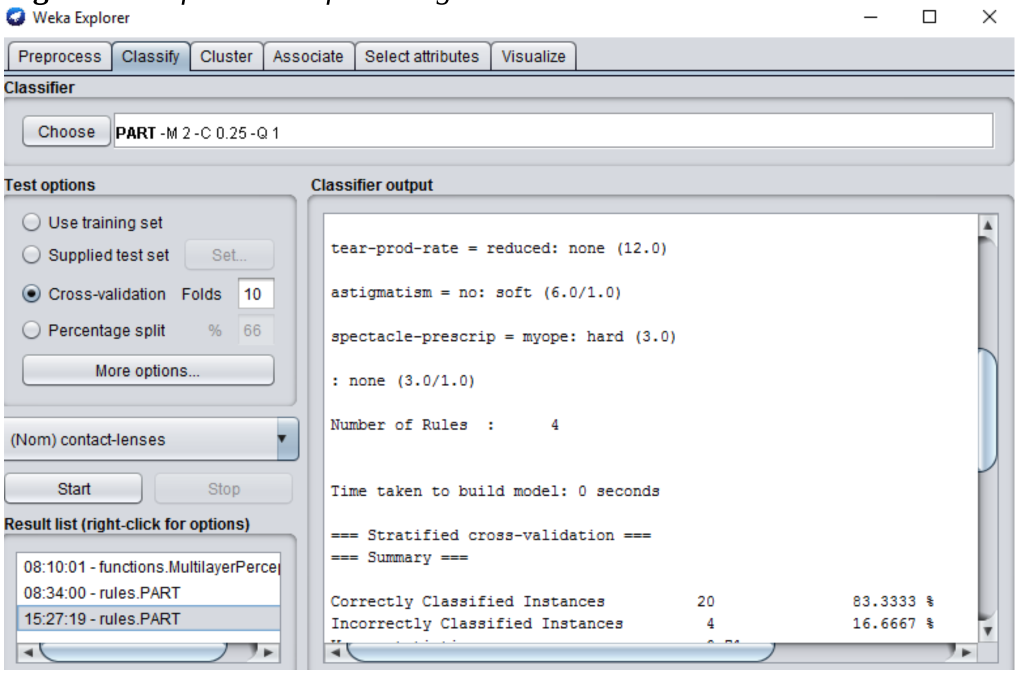
Results

Algorithms are executed in three datasets. The rules created by PART algorithm and the patterns found by Multilayer Perceptron are evaluated for accuracy. The datasets are partitioned in 10 subsets where 9 are used to find the rules and the patterns and one to test it for accuracy. The process is repeated in order to use all the 10 subsets as a test subset. The average of correctly classified observations on ten tests is displayed as the accuracy of algorithm.

The first dataset where the algorithms are executed is contact lenses data set.[9] The observations in this dataset are complete i.e. all possible combinations of attribute-value pairs are represented and correct. There are 24 observations. Each observation has 4 features: age of the patient, spectacle prescription, astigmatic and tear production rate and one conclusion: kind of contact lenses that should be taken.

The rules created by PART algorithm in Contact Lenses dataset were 83.33% accurate.

Figure 5. *Performance of Part Algorithm in Contact Lenses Dataset*



Classifier: PART -M 2 -C 0.25 -Q 1

Test options: Use training set; Supplied test set; [✓ Cross-validation] Folds 10; Percentage split % 66

(Nom) contact-lenses

Result list (right-click for options)
08:10:01 - functions.MultilayerPercep
08:34:00 - rules.PART
15:27:19 - rules.PART

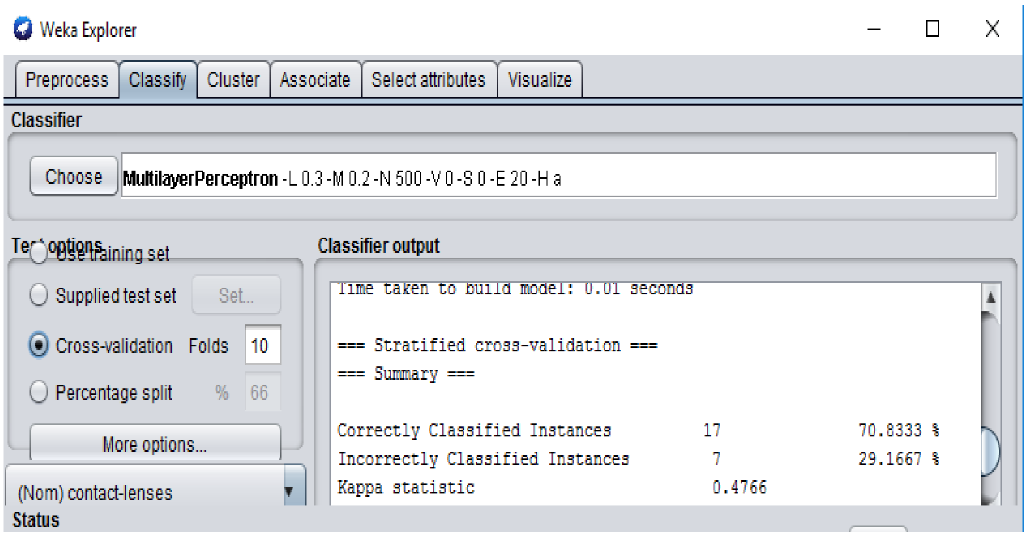
Classifier output

```
tear-prod-rate = reduced: none (12.0)

astigmatism = no: soft (6.0/1.0)

spectacle-prescrip = myope: hard (3.0)

: none (3.0/1.0)

Number of Rules  :      4


Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          20               83.3333 %
Incorrectly Classified Instances         4               16.6667 %
```

The patterns created by Multilayer Perceptron in contact lenses dataset was 70.8 % accurate.

[9]http://archive.ics.uci.edu/ml/datasets/Lenses.

Figure 6. *Performance of Multilayer Perceptron Algorithm in Contact Lenses Dataset*



Classifier

MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a

Test options

Use training set [✓ Cross-validation Folds 10] Supplied test set Percentage split % 66

(Nom) contact-lenses

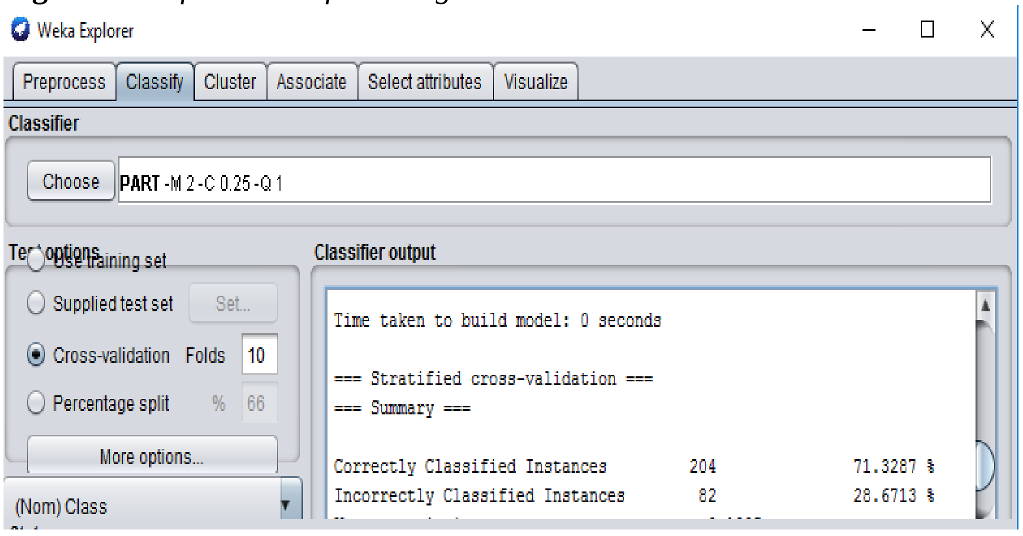
Classifier output

```
Time taken to build model: 0.01 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          17               70.8333 %
Incorrectly Classified Instances         7               29.1667 %
Kappa statistic                          0.4766
```

Another dataset that will be used for experiments is Breast Cancer dataset.[10] This data set includes 201 instances of one class and 85 instances of another class. The instances are described by 9 attributes, including age, tumor size, invasion nodes etc., some of which are linear and some are nominal. The conclusion (class) is about the recurrence of the tumor. There are 286 observations in this dataset. It is provided by the Oncology Institute and has repeatedly appeared in the machine learning literature.

The rules created by PART algorithm in Breast Cancer dataset were 71.3% accurate.

Figure 7. *Performance of Part Algorithm in Breast Cancer Dataset*



Classifier

PART -M 2 -C 0.25 -Q 1

Test options

Use training set [✓ Cross-validation Folds 10] Supplied test set Percentage split % 66

(Nom) Class

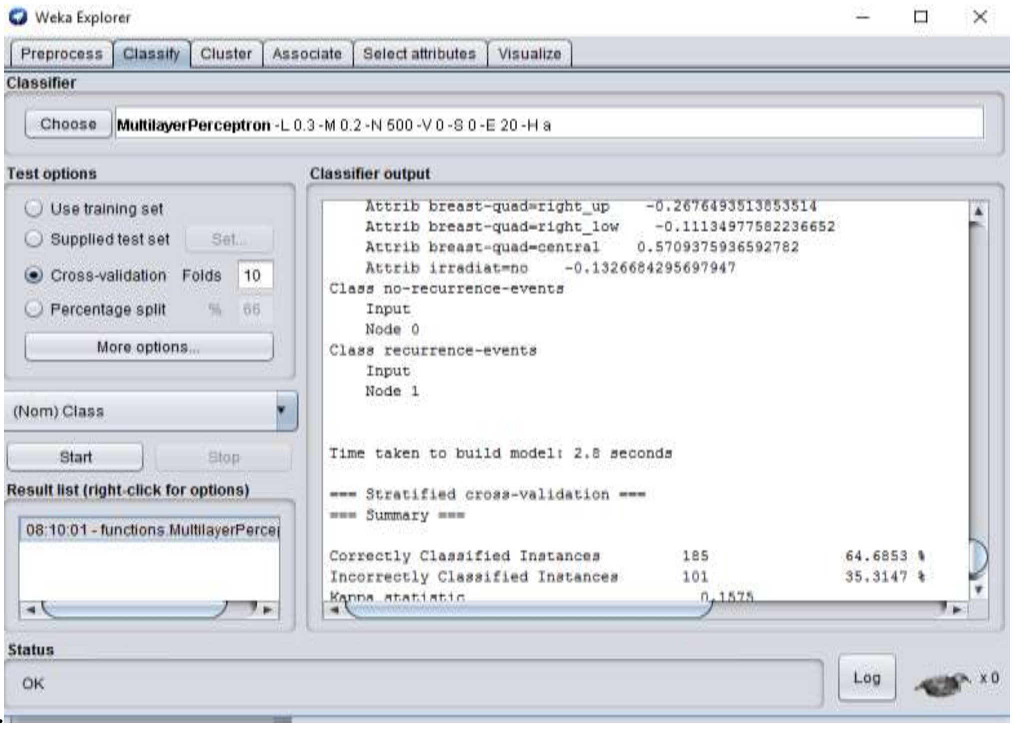
Classifier output

```
Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         204               71.3287 %
Incorrectly Classified Instances        82               28.6713 %
```

[10]https://archive.ics.uci.edu/ml/datasets/breast+cancer.

The patterns created by Multilayer Perceptron in Breast Cancer dataset was 64.68 % accurate.

Figure 8. *Performance of Multilayer Perceptron Algorithm in Breast Cancer Dataset*



MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a

Cross-validation Folds 10

(Nom) Class

08:10:01 - functions.MultilayerPerce

Classifier output

```
    Attrib breast-quad=right_up    -0.2676493513853514
    Attrib breast-quad=right_low    -0.11134977582236652
    Attrib breast-quad=central    0.5709375936592782
    Attrib irradiat=no    -0.1326684295697947
Class no-recurrence-events
    Input
    Node 0
Class recurrence-events
    Input
    Node 1


Time taken to build model: 2.8 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         185               64.6853 %
Incorrectly Classified Instances       101               35.3147 %
Kappa statistic                          0.1575
```

Status: OK

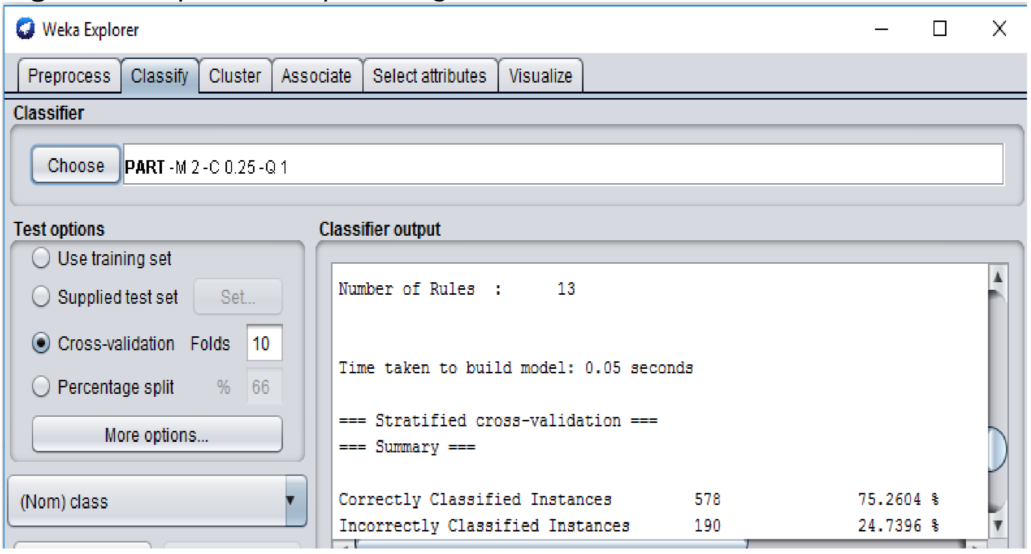
.

The last dataset that is used for experiments is Pima Indians Diabetes dataset.[11] All observations in this dataset are from females at least 21 years old of Pima Indian heritage. This dataset has 768 observations.

The rules created by PART algorithm in Pima Indians Diabetes dataset were 75.26% accurate.

[11]https://www.kaggle.com/uciml/pima-indians-diabetes-database.

Figure 9. *Performance of Part Algorithm Pima Indians Diabetes Dataset*



PART -M 2 -C 0.25 -Q 1

Cross-validation Folds 10

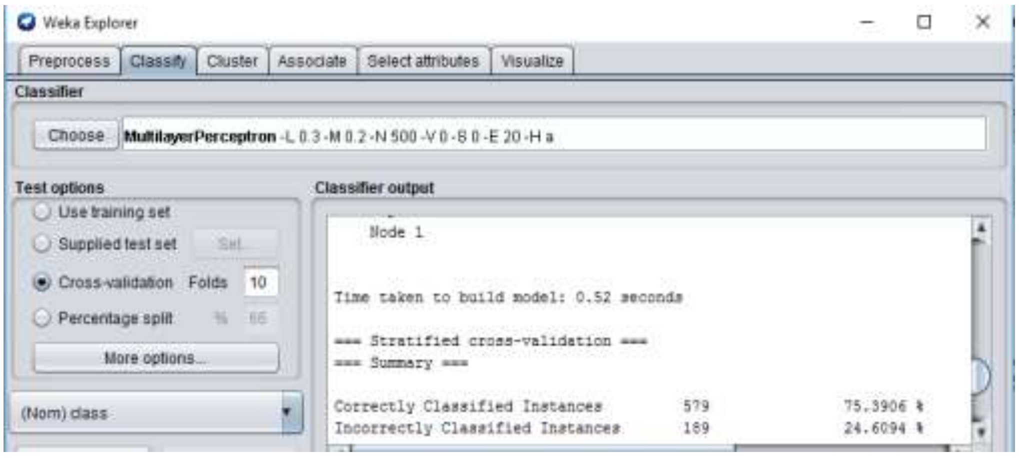
% 66

(Nom) class

```
Number of Rules  :      13


Time taken to build model: 0.05 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         578               75.2604 %
Incorrectly Classified Instances       190               24.7396 %
```

The patterns created by Multilayer Perceptron in Pima Indians Diabetes dataset was 75.39 % accurate.

Figure 10. *Performance of Multilayer Perceptron Algorithm in Pima Indians Diabetes Dataset*



MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a

Cross-validation Folds 10

% 66

(Nom) class

```
    Node 1


Time taken to build model: 0.52 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         579               75.3906 %
Incorrectly Classified Instances       189               24.6094 %
```

As it is seen from the experiments the PART algorithm, a representative algorithm for Knowledge Base CDSS, performs better in datasets with a small number of observations. As the number of observations is increasing the performance of PART algorithm is decreasing. On the other hand the Multilayer Perceptron, a representative algorithm for Non Knowledge Base CDSS, has a better performance in the dataset with the biggest number of observations. A huge amount of data is generated in health care organizations. As a result Non Knowledge Base CDSS will perform better than Knowledge Base CDSS in medicine area.

Conclusions

Clinical Decision Support Systems (CDSS) are developed in medicine area to address a necessity: convert raw produced data into useful information that will support the diagnosis and/or treatment decision making. There are two well-known types of CDSS: Knowledge Based and Non Knowledge Based CDSS. There is a considerable number of Knowledge-Based CDSS implementation in the medicine field due to the intuitive perception that these systems improve healthcare quality. It is widely believed that medicine profit more than the other fields from these systems. The rules in Knowledge Based CDSS are easy to understand and allow documentation of knowledge, self-learning, explanation, intelligent support in decision making and reasoning. On the other hand the development of Knowledge Based CDSS is very time consuming and their accuracy is decreasing with the increase of observations' numbers (data records). The last one is a big disadvantage. Healthcare organizations generate a tremendous amount of data. Experiments show that Non-Knowledge Based systems perform better in big datasets. Now that we are in Big Data Era Non-Knowledge Based CDSS will perform better than Knowledge Based CDSS.

References

Angela L. P. Chow, David C. Lye and Onyebuchi A. Arah (2015) Mortality Benefits of Antibiotic Computerized Decision Support System: Modifying Effects of Age; Scientific Reports volume 5, Article number: 17346. Doi: 10.1038/srep17346.

Chapman and Hall/CRC (2001) The Practical Handbook Of Genetic Algorithms Applications.

Clemens Scott Kruse, Amanda Beane (Published online 2018 Feb 5) Health Information Technology Continues to Show Positive Effect on Medical Outcomes: Systematic Review; J Med Internet Res. 2018 Feb; 20(2): e41. Doi: 10.2196/jmir.8793; PMCID: PMC5818676.

Garg A. X., Adhikari N. K., McDonald H., Rosas-Arellano M. P., Devereaux P. J., Beyene J., et al. (2005) "Effects of computerized clinical decision support systems on practitioner performance and patient outcomes: a systematic review." JAMA. 293 (10): 1223-38. PMID: 15755945DOI: 10.1001/jama.293.10.1223.

Goldberg, David (1989) Genetic Algorithms in Search,Optimization and Machine Learning. *Reading, MA: Addison-Wesley Professional.* ISBN 978-0201157673.

Nachtigall I., Tafelski S., Deja M., et al. Long-term effect of computer-assisted decision support for antibiotic treatment in critically ill patients: a prospective 'before/after' cohort study. BMJ Open 2014;4:e005370. Doi:10.1136/bmjopen-2014- 005370.

Quinlan, J. R. (1979) Discovering rules by induction from large collections of examples. In D. Michie (Ed.), Expert systems in the micro electronic age. Edinburgh University Press.

Quinlan, J. R. (1983) Learning efficient classification procedures and their application to chess endgames. In R. S. Michalski, J. G. Carbonell & T. M. Mitchell, (Eds.), Machine learning: An artificial intelligence approach. Palo Alto: Tioga Publishing Company.

Russell, Stuart J., Peter Norvig (2016) Artificial Intelligence: Pearson New International Edition: A Modern Approach Paperback. Pearson Education Limited.
Tanveer Syeda-Mahmood (March 2015) plenary talk: "The Role of Machine Learning in Clinical Decision Support". *SPIE Newsroom*. Doi:10.1117/2.3201503.29.
Yegnanarayana, B. (2009) Artificial Neural Networks. PHI Learning Pvt. Ltd.
Ying-Hong Liao, Chuen-Tsai Sun (2001) An Educational Genetic Algorithms Learning Tool. Retrieved on March 2018 from: https://bit.ly/2Ome5pm.

Annex A

```
package com.dt.ml;
importjava.util.ArrayList;
public class DecisonTreeCons {
        ArrayList<children> child;
        public class children{
                DecisonTreeConschildPointer;
                int value;
                children(DecisonTreeCons address, int v){
                        value = v;
                        childPointer = address;
                }
        }
        ArrayList<Record> data;
        double entropy;
        ArrayList<Integer>remainingfeatures;
        intbestfeatureindex;
        intpos_classcount;
        intneg_classcount;
        booleanisLeaf;
        String classvalue, className;
        HW1 mainClass = new HW1();

        publicDecisonTreeCons(){
        }
        publicDecisonTreeCons(ArrayList<Record>        records,        ArrayList<Integer>
remainingfeatures1){
                data = records;
                remainingfeatures = new ArrayList<Integer>();
                for(int count = 0; count < remainingfeatures1.size(); count++){
                        this.remainingfeatures.add(remainingfeatures1.get(count));
                }
                pos_classcount = calc_poscount(data);//calculates # of positive class labels
for a set of records
                neg_classcount = records.size() - pos_classcount;
                entropy = calcEntropy(data);
                isLeaf = false;
                child = new ArrayList<DecisonTreeCons.children>();
```

```
	}
	public void buildTree() {
		if(remainingfeatures.size() == 0 ){
			setLeaf(1); //Leaf Node-- remaining features are same
		}
		else if(pos_classcount == data.size() || neg_classcount == data.size()){
			setLeaf(2); //Leaf -- class values are same
		}
		else{
			bestfeatureindex = findBestFeature();
			if(bestfeatureindex != -1){
				//System.out.println("Defining feature is: " +
remainingfeatures.get(bestfeatureindex));
				int temp = remainingfeatures.get(bestfeatureindex);
				className =
mainClass.getFeatureName(remainingfeatures.get(bestfeatureindex));
				remainingfeatures.remove(bestfeatureindex);
				for(int j = 0; j< 2;j++){
					ArrayList<Record>subrecordschild =
getSubrecord(data, temp, Integer.toString(j));
					if(subrecordschild.size() != 0){
						DecisonTreeConschild_temp = new
DecisonTreeCons(subrecordschild, remainingfeatures);
						child.add(new children(child_temp, j));
						child_temp.buildTree();
					}
					else{
						;//System.out.println(j + " this values has
0 records");
					}
				}
			}
			else{
				setLeaf(1);//Leaf -- Information gain is zero
			}
		}
	}
	public void setLeaf(int i) {
		if(i == 2){
			isLeaf = true;
			classvalue = data.get(0).class_label;
		}
		else if( i == 1){
			isLeaf = true;
			if(pos_classcount>neg_classcount){
				classvalue = "1";
			}
			else{
```

```
                        classvalue ="0";
                    }
                }
        }
        publicintfindBestFeature() {
                doublemax_infogain = 0;
                int index =  0;
                for(int i = 0; i<remainingfeatures.size();i++){
                    doublesub_pos_neg_entropies = 0;
                    for(int j=0; j<2; j++){
                        ArrayList<Record>subdata        =        getSubrecord(data,
remainingfeatures.get(i), Integer.toString(j));
                        double  curvalue_ent  =  findCurrentEnt(subdata);
                        sub_pos_neg_entropies                    +=              (
(double)(double)subdata.size()/(double)data.size() )* curvalue_ent;
                    }
                    ;
                    if(max_infogain< (entropy - sub_pos_neg_entropies)){
                        max_infogain = entropy - sub_pos_neg_entropies;
                        index = i;
                    }

                }
                if(max_infogain == 0){
                    return -1;//  zero info gain
                }
                return index;
        }

        private double findCurrentEnt(ArrayList<Record>subdata) {
                doublenegativeent = 0;
                doublepositiveevnt =  0;
                for(int i = 0 ; i<subdata.size();i++){
                    if(subdata.get(i).class_label.equals("0")){
                        negativeent += 1;
                    }
                    else{
                        positiveevnt += 1;
                    }
                }
                doublesubdatasize = subdata.size();
                double a = -plogp(positiveevnt/subdatasize);
                double b = -plogp(negativeent/subdatasize);
                return (a+b);
        }

        publicArrayList<Record>getSubrecord(ArrayList<Record>datalist, intfeatureposition,
String featurevalue) {
		//returns array list of records having particular feature - indicated by feature
position -- as feature value - 0 or 1
		ArrayList<Record> subset = new ArrayList<Record>();
		for(int i = 0; i <datalist.size(); i++) {
			Record record = datalist.get(i);
			if(record.getRecord_row()[featureposition].equals(featurevalue)) {
				subset.add(record);
			}
		}
		return subset;
	}

	public double calcEntropy(ArrayList<Record> records){
		if(records.size() == 0)
			return -1;
		intposcount_class = 0;
		intnegcount_class = 0;
		doublerecsize = records.size();
		for(int j = 0; j <records.size(); j++) {
			Record record = records.get(j);
			int a = Integer.parseInt(record.getClass_label());
			if(a == 1) {
				poscount_class += 1;
			}
			else{
				negcount_class += 1;
			}
		}
		doublepos_prob = (double)poscount_class/(double)recsize;
		doubleneg_prob = (double)negcount_class/(double)recsize;
		doubleent =  -plogp(pos_prob) - plogp(neg_prob);
		returnent;
	}
	private static double plogp(double value) {
		if ( value == 0 ){
			return 0;
		}
		double a =  value *( Math.log(value) / Math.log(2));
		if(!Double.isNaN(a)){
			return a;
		}
		else{
			return 0;
		}
	}
	publicintcalc_poscount(ArrayList<Record> records) {
		//calculates # of positive class labels for a set of records
		int count = 0;
```

```
        for(int j = 0; j <records.size(); j++) {
            Record record = records.get(j);
            int a = Integer.parseInt(record.getClass_label());
            if(a == 1) {
                count += 1;
            }
        }
        return count;
    }
    public String printDtree(inttabCount) {
        tabCount++;
        if(isLeaf ){
            returnclassvalue;
        }
        else{
            for(int counter = 0; counter <child.size(); counter++ ){
                System.out.println();
                for(int count = 0; count <tabCount; count++){
                    System.out.print("| ");
                }
                System.out.print("|" + className + "= " +
child.get(counter).value + ": ");
                String formatPrint =
child.get(counter).childPointer.printDtree(tabCount);
                if(formatPrint.equals("0") || formatPrint.equals("1")){
                    System.out.print(formatPrint);
                }
            }
            return "null";
        }
    }
    public String traverseTree(Record testRecord){
        DecisonTreeCons node = this;
        while(node.isLeaf != true ){
            String best_feature_name = node.className;
            intbest_feature_index =
HW1.printfeatures.indexOf(best_feature_name);
            inttestrec_value =
Integer.parseInt(testRecord.getRecord_row()[best_feature_index]);
            node = node.child.get(testrec_value).childPointer;
        }
        String obtained_classlabel = node.classvalue;
        returnobtained_classlabel;
    }
}
```

Annex B

```
importjava.util.ArrayList;
importjava.util.Random;
import java.io.*;


public class  MultilayerPerceptron {

        // main constructor
        public  MultilayerPerceptron(intmp_neurons[])
        {
                Random rand = new Random();

                // create the required layers
                _layers = new ArrayList<Layer>();
                for (int i = 0; i <mp_neurons.length; ++i)
                        _layers.add(
                                        new Layer(
                                                        i == 0 ?
                                                        mp_neurons[i]  :mp_neurons[i  -
1],
                                                        mp_neurons[i], rand)
                                        );

                _delta_w = new ArrayList<float[][]>();
                for (int i = 0; i <mp_neurons.length; ++i)
                        _delta_w.add(new float
                                                [_layers.get(i).size()]
                                                [_layers.get(i).getWeights(0).length]
                                         );

                _grad_ex = new ArrayList<float[]>();
                for (int i =  0; i <mp_neurons.length; ++i)
                        _grad_ex.add(new float[_layers.get(i).size()]);
        }

        public float[] evaluate(float[] inputs)
        {
                // propagate the inputs through all neural network
                // and return the outputs
                assert(false);

                float outputs[] = new float[inputs.length];

                for(int i = 0; i < _layers.size(); ++i ) {
                        outputs = _layers.get(i).evaluate(inputs);
                        inputs = outputs;
```

```
		}

		return outputs;
	}

	private float evaluateError(float nn_output[], float desired_output[])
	{
		float d[];

		// add bias to input if necessary
		if (desired_output.length != nn_output.length)
			d = Layer.add_bias(desired_output);
		else
			d = desired_output;

		assert(nn_output.length == d.length);

		float e = 0;
		for (int i = 0; i <nn_output.length; ++i)
			e += (nn_output[i] - d[i]) * (nn_output[i] - d[i]);

		return e;
	}

	public float evaluateQuadraticError(ArrayList<float[]> examples,
								ArrayList<float[]>
results)
	{
		// this function calculate the quadratic error for the given
		// examples/results sets
		assert(false);

		float e = 0;

		for (int i = 0; i <examples.size(); ++i) {
			e += evaluateError(evaluate(examples.get(i)), results.get(i));
		}

		return e;
	}

	private void evaluateGradients(float[] results)
	{
		// for each neuron in each layer
		for (int c = _layers.size()-1; c >= 0; --c) {
			for (int i = 0; i < _layers.get(c).size(); ++i) {
				// if it's output layer neuron
				if (c == _layers.size()-1) {
```

```
                                        _grad_ex.get(c)[i] =
                                                2    *    (_layers.get(c).getOutput(i)    -
results[0])
                                                 *
_layers.get(c).getActivationDerivative(i);
                                }
                                else { // if it's neuron of the previous layers
                                        float sum = 0;
                                        for (int k = 1; k < _layers.get(c+1).size(); ++k)
                                                sum += _layers.get(c+1).getWeight(k, i)
* _grad_ex.get(c+1)[k];
                                        _grad_ex.get(c)[i]                              =
_layers.get(c).getActivationDerivative(i) * sum;
                                }
                        }
                }
        }

        private void resetWeightsDelta()
        {
                // reset delta values for each weight
                for (int c = 0; c < _layers.size(); ++c) {
                        for (int i = 0; i < _layers.get(c).size(); ++i) {
                                float weights[] = _layers.get(c).getWeights(i);
                                for (int j = 0; j <weights.length; ++j)
                                        _delta_w.get(c)[i][j] = 0;
            }
                }
        }

        private void evaluateWeightsDelta()
        {
                // evaluate delta values for each weight
                for (int c = 1; c < _layers.size(); ++c) {
                        for (int i = 0; i < _layers.get(c).size(); ++i) {
                                float weights[] = _layers.get(c).getWeights(i);
                                for (int j = 0; j <weights.length; ++j)
                                        _delta_w.get(c)[i][j] += _grad_ex.get(c)[i]
                                           * _layers.get(c-1).getOutput(j);
                        }
                }
        }

        private void updateWeights(float learning_rate)
        {
                for (int c = 0; c < _layers.size(); ++c) {
                        for (int i = 0; i < _layers.get(c).size(); ++i) {
                                float weights[] = _layers.get(c).getWeights(i);
```

```
                              for (int j = 0; j <weights.length; ++j)
                                  _layers.get(c).setWeight(i,                       j,
_layers.get(c).getWeight(i, j)
                                          -          (learning_rate            *
_delta_w.get(c)[i][j]));
            }
                }
        }

        private void batchBackPropagation(ArrayList<float[]> examples,

        ArrayList<float[]> results,

        floatlearning_rate)
        {
                resetWeightsDelta();

                for (int l = 0; l <examples.size(); ++l) {
                        evaluate(examples.get(l));
                        evaluateGradients(results.get(l));
                        evaluateWeightsDelta();
                }

                updateWeights(learning_rate);
        }

        public void learn(ArrayList<float[]> examples,
                                        ArrayList<float[]> results,
                                        floatlearning_rate)
        {
                // this function implements a batched back propagation algorithm
                assert(false);

                float e = Float.POSITIVE_INFINITY;

                while (e > 0.001f) {

                        batchBackPropagation(examples, results, learning_rate);

                        e = evaluateQuadraticError(examples, results);
                }
        }

        privateArrayList<Layer> _layers;
        privateArrayList<float[][]> _delta_w;
        privateArrayList<float[]> _grad_ex;
        /**
         * @paramargs
         */
        public static void main(String[] args) {

                // initialization
                ArrayList<float[]> ex = new ArrayList<float[]>();
                ArrayList<float[]> out = new ArrayList<float[]>();
                for (int i = 0; i < 4; ++i) {
                        ex.add(new float);
                        out.add(new float);
                }

                // fill the examples database
                ex.get(0)[0] = -1; ex.get(0) = 1;  out.get(0)[0] = 1;
                ex.get(1)[0] = 1;  ex.get(1) = 1;  out.get(1)[0] = -1;
                ex.get(2)[0] = 1;  ex.get(2) = -1; out.get(2)[0] = 1;
                ex.get(3)[0] = -1; ex.get(3) = -1; out.get(3)[0] = -1;

                intmp_neurons[] = {
                                ex.get(0).length,   // layer 1: input layer - 2 neurons
                                ex.get(0).length * 3,       //  layer  2:  hidden  layer  -  6
neurons
                                1                           //  layer  3:  output  layer  -  1
neuron
                };

                MultilayerPerceptronMultilayerPerceptron            =            new
MultilayerPerceptron(mp_neurons);

                try {
                        PrintWriterfout = new PrintWriter(new FileWriter("plot.dat"));
                        fout.println("#\tX\tY");

                        for (int i = 0; i < 40000; ++i) {
                                MultilayerPerceptron.learn(ex, out, 0.3f);
                                float                   error                   =
MultilayerPerceptron.evaluateQuadraticError(ex, out);
                                System.out.println(i + " -> error : " + error);
                                fout.println("\t" + i + "\t" + error);
                        }

                        fout.close();
                } catch (IOException e){
                        e.printStackTrace();
                }
        }
}
```