**CSC3423 Bio-Computing**

**Coursework: Biologically-inspired computing for machine learning**

[Introduction 1](#_Toc496092932)

[Specification of “coursework proposal” 2](#_Toc496092933)

[Specification of “practical report” 2](#_Toc496092934)

[Description of the machine learning framework 3](#_Toc496092935)

[Sample solution 5](#_Toc496092936)

[Example dataset and how to run experiments. 6](#_Toc496092937)

# Introduction

This document describes the specification for **both** pieces of coursework of CSC3423, identified in NESS as “coursework proposal” and “practical report”.

*Submission will be via the NeSS system https://ness.ncl.ac.uk*

The learning objectives of CSC3423’s coursework are:

1. To understand how biologically-inspired methods can be applied to the specific computational task of machine learning
2. To critically evaluate the suitability of different types of biologically-inspired computing for a given task
3. To practice the skill of scientific experimental reporting
4. To learn how to integrate one’s code within a given code base

The objective of the coursework is to use biologically-inspired computing for machine learning. You will be provided with the (Java) source code of a general framework for machine learning (described below) in which you will be asked to (1) select two different types of biologically-inspired computing and integrate (and tune) them within this framework to train classifiers and (2) critically assess and compare the suitability of the methods you have selected and implemented. Two datasets to perform your experiments are also provided.

A sample implementation of the coursework using a Genetic Algorithm as the biologically-inspired algorithm of choice is provided. **Submissions that are too similar to this example will obtain a very low mark.**

Notes:

- Early in the lectures of the module there will be an introduction to machine learning and an in-depth description of the coursework

- You are free to implement the whole code of the biologically-inspired algorithms or integrate any of the numerous existing open source implementations of these methods into the provided machine learning framework

- You should not modify the provided framework except in the places where you are explicitly told to do so. If you really feel the need to modify the code, please contact the lecturer **before** the submission to seek permission.

- Your code should not make any assumption about the characteristics of the data. Use the provided API to determine the structure of the problem. That is, make your code generic. Given that the same code has to work for the two datasets (that have different number of attributes), this is a must.

- For both pieces of assessment it is VERY important that you use references and citations correctly and do not copy material from other sources without correctly citing it. Please see the University information on plagiarism.

# Specification of “coursework proposal”

For this piece of assessment you have to write a report where you explain your proposal of design for the coursework. That is, for each of the two chosen biologically-inspired algorithms, you have to explain (1) why you think it is suitable for the coursework’s machine learning task and (2) your strategy for integrating such algorithm within the provided software framework, including the choice of knowledge representation used to tackle the machine learning task.

This assignment will comprise 10% of the total marks of the module.

Marks will be awarded for:

* Justification for the selected nature-inspired algorithms: 30 marks
* Description of the knowledge representation: 20 marks
* Software design for the integration of the nature-inspired algorithms within the provided framework: 30 marks
* Figures: 10 marks
* Writing: 10 marks

The report has a word limit of **1000 words**.

# Specification of “practical report”

The second piece of assessment will focus on the full implementation, testing and critical evaluation of your proposal of using nature-inspired algorithms for the provided machine learning task. You are allowed to diverge from the design previously submitted in the “project proposal” assessment.

You have to submit (through NESS) the following two items:

* The zipped complete source code of your program (including the provided framework and your own code)
* A report (of max. **2000 words**) in which you explain, for each of the two selected nature-inspired algorithms, (a) why you selected it, (b) how you integrated it into the framework, (c) how you tuned it for the provided datasets and (d) its performance for the provided dataset in terms of (1) iterations of the nature-inspired algorithm that it takes to generate a classifier (2) predictive power of the classifier on training/test data and (3) run-time of the training proces. Also, critically evaluate and compare the suitability and performance of the selected nature-inspired methods

Marks will be awarded for:

* Revised justification and design for the selected nature-inspired algorithms and knowledge representations: 20 marks
* Description of your implementation: 30 marks
* Description of how your implementation was adjusted and tuned for the provided dataset: 10 marks
* Report of performance: 20 marks
* Critical comparison between methods and overall reflection: 20 marks
* **BONUS**: Generate visualisations of the solutions of your method and/or the functioning of your implementation of the nature-inspired methods: Up to 10 marks.

This assignment will comprise 40% of the total marks for the module.

# Description of the machine learning framework

**- NOTE:** Before reading the text below, please be familiar with all the machine learning nomenclature that is explained in “coursework preparation lecture” of the module.

**-** You can download the machine learning framework from Blackboard at “Teaching Materials > Assessment”.

You are provided with a (minimalistic) machine learning framework in Java. The framework is composed of 11 Java classes:

* Attribute.java. This class holds the characteristics of each attribute in the dataset. From its functions the parts that are important to you are:
  + The name of the attribute – getName()
  + The minimal and maximal value of the domain of the attribute; minAttribute() and maxAttribute()
* Attributes.java. This class contains information about the metadata of the dataset: the number of attributes, the characteristics of each attribute and the number of classes. Important parts of this class for you are:
  + The number of attributes in the dataset ; getNumAttributes()
  + Fetching the Attribute object corresponding to each attribute; getAttribute(int pos)
  + Getting the number of classes in the problem: public attribute numClasses
* Classifier.java. **This is one of the most important classes for you**. It contains the interface for the classifiers you will have to generate (using biologically-inspired methods).
  + You will have to implement the following methods:
    - public abstract int classifyInstance(Instance ins). This function will receive an instance and attemps to predict its class. If the classifier can predict its class it will return a class index (between 0 and the number of classes in the dataset-1). If it cannot predict this instance it will return -1
    - public abstract void printClassifier(). Prints to screen a textual description of the classifier
  + Moreover, you are provided with three auxiliary functions.
    - public double getFitness(). This function will return the last fitness value that was computed for this classifier
    - public void computeFitness(InstanceSet is). This function will receive a training set and will compute the fitness of the classifier. That is, how good is at predicting all/part of the instances of the training set. You can use this function to evaluate the solutions while they are generated by any of the population-based biologically-inspired algorithms (e.g. genetic algorithms, genetic programming, memetic algorithms, ant colony optimisation, particle swarm optimisation).
    - public void computeStats(InstanceSet is). This function will print to screen a few performance statistics of the classifier given a dataset
* ClassifierAggregated.java. This class contains a collection of classifiers that together are a complete solution to the classification problem. That is, (should) cover the whole space of solutions. **You don’t have to modify/access this class directly**
* Control.java. This is the main class of the framework that controls the training process.
  + The main function implements the learning algorithm generally known as *iterative rule learning*, that iteratively builds a solution (*ClassifierAggregated* object) by sequentially learning classifiers (that implement *Classifier* interface).
  + To learn each classifier the generateSubsolution(InstanceSet trainingSet) function is called. **Here is where you have to put the code of the biologically-inspired algorithm** to train a classifier using *trainingSet* and return a *Classifier* object.
* Instance.java. This class represents each of the instances (i.e. data points, database rows) of the dataset. Important functions to you are:
  + public double getRealAttribute(int attr). Return the value of this instance for attribute *attr*
  + public int getClassValue(). Returns the class label of this instance
* InstanceSet.java. This object holds a complete instance set (a collection of instances). Functions important to you are:
  + public int numInstances(). Returns the number of instances of the instance set
  + public Instance getInstance(int whichInstance). Return an *Instance* object for instance with index *whichInstance*
  + public Instance[] getInstances(). Returns all instances in the object as an array
* MTwister.java. This class implements the *Mersenne Twister* pseudo-random number generator (PRNG). You don’t need to access this file.
* ParserARFF.java. This class reads the file format (called ARFF) used to specify the dataset. You don’t need to access this file.
* Rand.java. This class is the interface to the pseudo-random number generator. Four funtions are important for you:
  + public static void initRand(). This function initialises the generator using the standard Java PRNG. This is the function called from Control.java. Hence, each time that you call the program the PRNG will generate different numbers.
  + public static void initRand(long seed). Initialises the PRNG with a specific seed. This means that the program will always provide the same output if it is specified with the same seed.
  + public static double getReal(). Returns a real number between [0,1]
  + public static int getInteger(int uLow, int uHigh) . Returns an integer number betwee *uLow* and *uHigh.*

# Sample solution

The GAwrapper.java and ClassifierSphere.java classes contain a sample implementation of the coursework using a Genetic Algorithm and an open source library called jenetics (http://jenetics.io/).

The genetic algorithm is designed to optimise classifiers that are defined as a sphere. Each sphere is defined by (a) a center (b) a radius and (c) a class associated to the sphere. The classifier will be activated by any instance for which the euclidean distance with the center is less than the radius, and will predict that it belongs to its associated class. The class ClassifierSphere.java implements this type of knowledge representation. The fitness function it implements is a wrapper for the fitness function provided by the machine learning framework.

The GAwrapper.java is a relatively short file (less than a hundred lines of code in total) that specifies how to set up a genetic algorithm using the jenetics library and how to return objects following the Classifier interface to the coursework’s control class.

The class has three main functions:

1. evaluate. This is the function that the jenetics framework will call to evaluate the individuals of the GA population. It receives an individual of ‘Genotype<DoubleGene>’ type and returns a double. It makes use of the next function to build and evaluate a classifier from the genotype provided as argument to the function.
2. buildClassifier. This function receives a ‘Genotype<DoubleGene>’ object and builds a Classifier (class from the provided coursework framework) from the information encoded in such genotype. This function is an essential piece of the glue connecting the coursework framework to the jenetics library, as it makes the mapping from the basic individuals evolved by jenetics to the Classifiers that the provided coursework framework requires. If you look at the code you will see that the individuals evolved by jenetics multiplex the information of the sphere as a single vector containing, in this order, the coordinates of the centre of the sphere, its radius and its associated class. The associated class is defined as a continuous variable in the Genotype. To convert it to an integer we simply truncate the decimals. This is a quick fix for the fact that jenetics prefers to encode genotypes where all genes have the same type.
3. generateClassifier. This is the main function of this class. It is in charge of setting up and run the genetic algorithm using jenetics. The function has three main blocks: (1) defining the genotype. (2) setting up the components of the genetic algorithm and (3) running it. For part (1) the important aspect to remember is that the code cannot make assumptions about the number of attributes or the number of classes (the *machine learning* meaning of ‘class’) in the dataset. It needs to ask this information to the Attribute/Attributes classes (the *programming* meaning of ‘class’). For part (2), you will see in the code how the GA is parametrised: population size, number of elites, selection/crossover/mutation operators and parameters. For part (3), the code specifies the number of generations of the GA, sets the algorithm to run, collects the best individual from the population and returns a Classifier object made from that individidual.

# Example dataset and how to run experiments.

For this coursework you are provided with two dataset that represents the “Ying-Yang” figure. For each dataset you will have two files: a training set and a test set. (dataset1-Training.arff, dataset1-Test.arff, dataset2-Training.arff, dataset2-Test.arff).

plot-dataset.pdfIn dataset 1 each instance has two attributes, x and y and a class with values “black” and “white”. If you plot the instances as if they were coordinates you would see this:

Dataset 2 is a version of dataset 1 in which each row has four extra columns added to it. In reality these four columns are irrelevant, they are not linked to the class attribute. **However, you code should not incorporate this knowledge, as it cannot make any assumption on the data.**

If you look at the Control.java file you will see that the program receives two arguments. The first one is the training file and the second one is the test file.

Here is an example output of the framework for the sample implementation (The output depends on the random seed) when calling the code from the UNIX command line:

Random seed is 1152352159224889151

Relation name D14

Attribute name att15

Attribute name att16

Attribute name class

Classifier of iteration 0. Accuracy 100.00%, coverage 26.11%

Iteration 0, removed 235 instances, instances left 665

Overall stats at iteration 0. Accuracy 26.11%, error rate 0.00%, not classified 73.89%

Classifier of iteration 1. Accuracy 100.00%, coverage 10.44%

Iteration 1, removed 84 instances, instances left 581

Overall stats at iteration 1. Accuracy 35.44%, error rate 0.00%, not classified 64.56%

Classifier of iteration 2. Accuracy 100.00%, coverage 2.56%

Iteration 2, removed 23 instances, instances left 558

Overall stats at iteration 2. Accuracy 38.00%, error rate 0.00%, not classified 62.00%

Classifier of iteration 3. Accuracy 100.00%, coverage 2.44%

Iteration 3, removed 9 instances, instances left 549

Overall stats at iteration 3. Accuracy 39.00%, error rate 0.00%, not classified 61.00%

Classifier of iteration 4. Accuracy 100.00%, coverage 11.33%

Iteration 4, removed 5 instances, instances left 544

Overall stats at iteration 4. Accuracy 39.56%, error rate 0.00%, not classified 60.44%

Classifier of iteration 5. Accuracy 100.00%, coverage 13.67%

Iteration 5, removed 3 instances, instances left 541

Overall stats at iteration 5. Accuracy 39.89%, error rate 0.00%, not classified 60.11%

Classifier of iteration 6. Accuracy 100.00%, coverage 1.11%

Iteration 6, removed 10 instances, instances left 531

Overall stats at iteration 6. Accuracy 41.00%, error rate 0.00%, not classified 59.00%

Classifier of iteration 7. Accuracy 100.00%, coverage 2.00%

Iteration 7, removed 18 instances, instances left 513

Overall stats at iteration 7. Accuracy 43.00%, error rate 0.00%, not classified 57.00%

Classifier of iteration 8. Accuracy 100.00%, coverage 2.00%

Iteration 8, removed 4 instances, instances left 509

Overall stats at iteration 8. Accuracy 43.44%, error rate 0.00%, not classified 56.56%

Classifier of iteration 9. Accuracy 100.00%, coverage 0.67%

Iteration 9, removed 6 instances, instances left 503

Overall stats at iteration 9. Accuracy 44.11%, error rate 0.00%, not classified 55.89%

Classifier of iteration 10. Accuracy 100.00%, coverage 0.44%

Iteration 10, removed 4 instances, instances left 499

Overall stats at iteration 10. Accuracy 44.56%, error rate 0.00%, not classified 55.44%

Classifier of iteration 11. Accuracy 100.00%, coverage 3.11%

Iteration 11, removed 3 instances, instances left 496

Overall stats at iteration 11. Accuracy 44.89%, error rate 0.00%, not classified 55.11%

Classifier of iteration 12. Accuracy 82.38%, coverage 58.00%

Iteration 12, removed 458 instances, instances left 38

Overall stats at iteration 12. Accuracy 92.67%, error rate 3.11%, not classified 4.22%

Classifier of iteration 13. Accuracy 95.32%, coverage 19.00%

Iteration 13, removed 9 instances, instances left 29

Overall stats at iteration 13. Accuracy 93.67%, error rate 3.11%, not classified 3.22%

Classifier of iteration 14. Accuracy 5.78%, coverage 19.22%

Iteration 14, removed 4 instances, instances left 25

Overall stats at iteration 14. Accuracy 94.11%, error rate 3.11%, not classified 2.78%

Classifier of iteration 15. Accuracy 58.16%, coverage 15.67%

Iteration 15, removed 2 instances, instances left 23

Overall stats at iteration 15. Accuracy 94.33%, error rate 3.11%, not classified 2.56%

Classifier of iteration 16. Accuracy 66.47%, coverage 75.22%

Iteration 16, removed 22 instances, instances left 1

Overall stats at iteration 16. Accuracy 96.44%, error rate 3.44%, not classified 0.11%

Classifier of iteration 17. Accuracy 49.44%, coverage 89.44%

Iteration 17, removed 1 instances, instances left 0

Overall stats at iteration 17. Accuracy 96.56%, error rate 3.44%, not classified 0.00%

Final classifier

cl0:Center 2.6217925581211325,0.9880713860694893, radius 0.40120828731237806, class 0

cl1:Center 2.0406576673328427,0.8715733052963621, radius 0.21861526967677955, class 0

cl2:Center 2.9496677658610997,0.6105989802023932, radius 0.10656894234625465, class 0

cl3:Center 2.519810085500992,0.6377269789044459, radius 0.09310618343169157, class 0

cl4:Center 2.8552823575019097,0.8358462791678762, radius 0.24017574483033235, class 0

cl5:Center 2.2591023370462553,0.9505395695809377, radius 0.2091731469276897, class 0

cl6:Center 2.5962678474013186,0.474477745202282, radius 0.059453999139511486, class 0

cl7:Center 2.2530295067138413,0.474159764663678, radius 0.08432162983798522, class 0

cl8:Center 2.2789553373753115,0.4737481841284768, radius 0.09206150351657623, class 0

cl9:Center 2.7382888159106087,0.31761981775768083, radius 0.047298627554258996, class 0

cl10:Center 2.0607478308964047,0.6237404811882252, radius 0.025541395504554143, class 0

cl11:Center 2.5588285311522405,0.6521783520091408, radius 0.10803356434636975, class 0

cl12:Center 2.3831337726965125,0.05796424721504286, radius 0.6406005271781253, class 1

cl13:Center 2.99573190465153,0.037187095478134514, radius 0.3835786021989401, class 1

cl14:Center 2.1993232149485085,0.8895084657376319, radius 0.24916674603785147, class 1

cl15:Center 2.5996314147758683,0.45266225542308697, radius 0.2388633045626901, class 1

cl16:Center 2.3074233219972062,0.9095934973200238, radius 0.804415843391279, class 0

cl17:Center 2.839520998646162,0.5720515283670136, radius 0.7716371032176577, class 1

Stats on test data

Accuracy 95.00%, error rate 5.00%, not classified 0.00%

Total time: 5.419

From this output there are a few things to note:

* The solution consisted in 18 spheres iteratively covering the instances of the training set. Please note that for the last iteration the number of instances left was 0. This is the end condition of the learning loop.
* The overall stats for the last iteration provide you with the performance measures of the collection of classifiers for the training data
* Afterwards, the classifier (the set of spheres) is printed.
* Next you have the stats on test data. A good classifier is one that is able to obtain good performance on the test data, as it shows that it is able to generalise from the training data.
* Finally, the last line tells you the run-time of the training and test process in seconds.