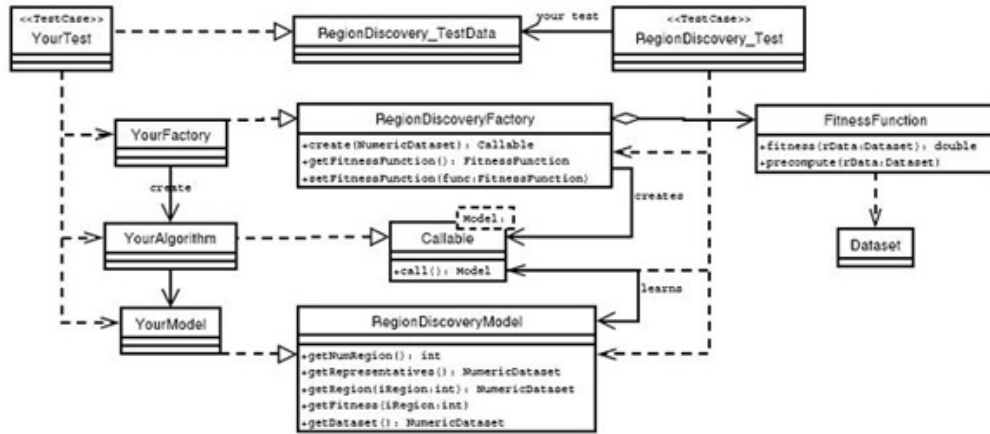


Figure 1: Class diagram for new software addition.



1 General Picture

In this part of the project, you will develop a new region discovery algorithm following the Test First Development approach. You will run experiments using your own algorithm.

2 Extending Cougar²

In this part you will add software to the framework. You will follow test first development. You will start with the design of your algorithm that fits into the hierarchy of figure 1. You can list all the unit tests required on a paper. Unit tests are small and each test tests just one aspect of the code. You should have more than 20 unit tests. Here are some guidelines about what kind of unit tests are expected:

- Parameter Checking
 - Test setting the fitness function. Use mock objects only. Do not use the fitness function from the previous assignment.
 - If your algorithm has parameters, you need to test getting and setting each parameter separately. Specify in your test the value you use as in: “Assert that beta is 0 by default. Set beta to 0.1, and assert that beta is now 0.1”.
 - Check parameter values in the constructor of your learner (Your Algorithm). Expect a CreationException if the parameters are inconsistent.
- Create Learner
 - Exceptions on invalid datasets such as dataset with 0 examples or 0 attributes.

- Model: After you create the learner and provide a dataset, the model should have several properties.
 - If your dataset should have 3 regions (each is one unit test after you build the model):
 - * Model has 3 regions
 - * Each example belongs to one of the three regions (if your algorithm does not have concept of outliers)
 - * All regions have at least one example.
 - * The number of representatives is 3 (in case of representative-based algorithm)
 - * The representatives are the right values.

After listing unit tests on the paper, start implementing your unit tests and not the actual code. Write just the necessary code so that unit tests compile. All the unit tests will fail at this point as there will be no code to make them pass. In the next step, start implementing your algorithm code. Some guidelines for writing the code are:

- Factories
 - Get and set methods handle all the parameters needed by the algorithms
 - A create method performs two kinds of checking:
 - * Parameters must be consistent. For example, if the number of clusters, k , is set, then $k < \text{Dataset size}$.
 - * Any datasets needed for training are defined appropriately. Any attributes you give to the factory must be Attribute objects and not indexes because you can not assume that the index matches the dataset that you provide. Remember that create method performs all these checks through the learner constructor.
 - * The create method ensures that the created object can be called without configuration errors.
 - Factories can be serialized and de-serialized.
- Learners: Each region discovery algorithm is a learner.
 - Learners implement a single method, call, that performs the learning and returns a model.
 - Learners do not allow the user to change any configuration parameters. For example, if a parameter is a double array, changes to the elements in the array do not in any way affect the learner; thus, the array should be copied from the factory to the learner.
 - There are no set methods in a learner. Get methods may exist, but must not return anything that could affect the results of a learner.
 - There is no hierarchy of learners, only factories and models.

- Models: Each region discovery algorithm will implement a region discovery model.
 - A model stores all data and functionality needed to describe and reuse the result of the algorithm.
 - A model may take parameters but changing the parameters must not require re-learning.
 - All visualization and analysis methods use only the model and a dataset.
 - Models can be serialized and reused.

Create a package for your algorithm in the `dmml.regionDiscovery` package in `src/main/java` folder. Create a package for your unit tests in `dmml.regionDiscovery` package in `src/test/java` folder. For your experiments, create necessary packages in `examples.experiment` package in `src/examples/java` folder.

3 Experiments using your algorithm

You will test your algorithm by running experiments with real datasets. See the instructions given for the previous assignment to run experiments. Run the experiment and analyze your results.

1. Run the algorithm with the purity fitness function ($\beta = 1.01$, Parameters for Purity: $\eta = 2$, $th = 0.6$) for the 9-Diamond dataset.
2. Run the algorithm with the variance fitness function ($\beta = 1.01$ and $\beta = 1.5$, Parameters for Variance: $\eta = 1$, $b = 10$) for the Arsenic dataset.
3. Run the algorithm with the purity fitness function ($\beta = 1.01$ and $\beta = 2.0$, Parameters for Purity: $\eta = 1$, $th = 0$) for the Volcano dataset. The dataset can be found at: <http://www.cs.uh.edu/~ykuo>

For experiment 1 and 2 visualize the results. Also visualize 3 regions with highest interestingness and 3 regions with highest reward and compare with the result obtained by CLEVER in part 2 of the project. In case of experiment 3, visualize the result and also the 3 regions with highest reward. Analyze all the experimental results and write a detailed report.

4 Submission

Email the following to ykuo@cs.uh.edu.

1. Submit three files of your algorithm, unit test file with failing unit tests and a status report by November 3, 11:59 PM CST. You need not have any code implemented other than the code necessary for compilation. In each unit test describe what you intend to test in the unit test. You can add more unit tests as you develop the code. As we follow Test First Development approach, first you are going to write tests.
2. Submit the following by November 19, 11:59 PM CST.

- Your code for factory, learner and model.
- Code of unit tests.
- Report describing what your classes do.
- Report describing your experimental results.
- Include your opinions in the report:
 - What did you like about the assignment?
 - What did you not like about the assignment?
 - What would you change about our software and why?

5 General Guidelines

- Observe the coding conventions used in the previous assignment
- If you do not know how to test something, look at the other tests. There are hundreds of unit tests already implemented in the source.
- Read the interface descriptions of classes if you are unsure how to use them.
- Do not assume that “The user is smart enough not to call some method or set some value”. If you expect the user to call some method, you should throw an exception if it was not called. If you expect some special values, you need to throw an exception if you did not get them.
- Make sure that your code compiles without errors and warnings. You will lose marks if your code has excessive amount of warnings.
- Start early. Send e-mail if you need help.