

## General Picture

In the first part of the project, you will get familiar with the Cougar<sup>2</sup> Framework and the employed software design approach. We will have two lab-style classes on September 4 and 6.

### 1 Getting Familiar with Cougar<sup>2</sup>

You will implement a new classifier called the RangeClassifier. Given an example  $x^n$ , the classifier performs the following classification:

$$h(x) = \begin{cases} 1 & x \in [x_{min}, x_{max}) \\ 0 & x \notin [x_{min}, x_{max}) \end{cases}$$

where  $x_1$  is the first attribute of  $x$ , and user-defined parameters  $x_{min} < x_{max} \in \mathbb{R}^n$  specify a range of allowable values. The class values indicate whether the first attribute of  $x$  is in range.

Although this is a simple classifier, programming it is a good demonstration of the issues we face in implementing data mining algorithms. In this assignment, we will use Test-First Development to develop a program for the classifier.

As part of the development process we will follow this steps:

1. Create the unit tests that specify core functionality before writing the real code.
2. Ensure that these tests fail (because you have not written the code yet).
3. Write code that makes the test pass.
4. Refactor the code so that it is more readable.

To create the unit test, we specify some behavior that is expected of the working code. Since we are writing the tests first, they should fail because the code does not exist. After ensuring that the code fails for the expected reasons, we then write code that makes the test pass. This helps us ensure that the code is well tested without relying on the traditional debugging and print statements.

In the `examples.rangeClassifier` package of the code repository, you will find skeleton code for the Classifier. The skeleton code contains all the methods you will need for the implementation. This code compiles and runs, but the unit tests fail.

For this assignment, you will complete the unit tests, one by one, that will cause the implementation to work as expected. For the test code, some code has been included that will help you write the tests.

The tests follow the convention in our framework that a classifier consists of 3 parts: Factory, Learner, Model. The factory configures the parameters and can be saved to a file to make several copies of the learner. A Learner builds a Model from the test data. A Classifier, which is a Learner, assumes that the data contains a class attribute that

has nominal values. A Model predicts values for the class attribute for a new dataset, which is assumed to have the same structure as the training dataset.

The unit tests for a learner specify what is expected of each component. Some examples of the unit tests you will write are:

- Factory
  - If the default range is used, is a valid learner created?
  - If the range is invalid, is a valid learner not created?
- Learner
  - Does the learner build a valid model given input data?
  - In this particular learner, we assert that the learner does not access the dataset at all.
- Model
  - Given the default range, does the model predict the right values?
  - Given a different range, does the model still predict the right values?
  - The model must not access the true class labels of the testing set.

In order to write the tests, you will follow a few rules:

- Do not create any new classes than what is provided.
- Do not implement any interface yourself.
- Do not access any code in the DMML package except what is already referenced in the imports.
- Do not share code with other people in the class.
- Do not include any print statements.
- You must use the EasyMock mock object library.
- All test cases must have at least one JUnit assert statement.

## 2 JUnit

As an example of the test cases you will implement, we will test that the RangeClassifierFactory returns default values of 0.0 and 0.5, respectively for the range variables. The test case should appear as follows:

```

@Before
public void setUp() throws Exception
{
    _testFactory=new RangeClassifierFactory();
}
@Test
public void defaultMinMax()
{
    assertEquals(0.5, _testFactory.getMax());
    assertEquals(0.0, _testFactory.getMin());
}

```

The bold text refers to structures from the JUnit library. The test assumes that that we have defined a variable named `_testFactory` with two methods: `getMax` and `getMin`. When this test is run, JUnit executes the method **setUp** before the **defaultMinMax** method. In the `setUp` method, we have initialized the `_testFactory` member variable. In a JUnit test, the test assumes that `setUp` is run before each test. The values returned by the methods should be equal to the stated values.

### 3 Setup

We will be using Eclipse IDE and our Cougar<sup>2</sup> software framework. These instructions should help you get started.

1. Join the project
  - (a) Register for an account at <http://www.java.net/>
  - (b) Add yourself to the project (<http://cougarsquared.dev.java.net/>) and request for the Observer role.
2. Installation: If you are using your portable computer, follow these steps. These steps will be demonstrated in lab-style classes.
  - (a) Install Java JDK 1.5 from <http://java.sun.com>
  - (b) Download and install Eclipse 3.2 or higher from <http://www.eclipse.org>.
  - (c) Install the subclipse plugin.
  - (d) In Eclipse, checkout project for the Cougar<sup>2</sup> Framework from <https://cougarsquared.dev.java.net/svn/cougarsquared/trunk/core>.
  - (e) Go to the code in the package “examples.rangeClassifier” in `src/examples/java` folder.
  - (f) Run the `RangeClassifierTest` code to verify that the tests fail at first.
3. For more information, see:
  - (a) <http://www.eclipse.org>

- (b) <http://www.easymock.org>
- (c) <http://www.junit.org>
- (d) <http://www.javaworld.com> for articles about Java programming.
- (e) <http://java.sun.com/j2se/1.5.0/docs/index.html> for Java 1.5.

## 4 Submission

E-mail the following to [ykuo@cs.uh.edu](mailto:ykuo@cs.uh.edu) (cc [rachana.parmar@gmail.com](mailto:rachana.parmar@gmail.com)) on September 13, 2007:

1. Subject: COSC 6397: Project
2. 4 Source files: `RangeClassifier.java`, `RangeClassifierFactory.java`, `RangeClassifierModel.java`, and `RangeClassifierTest.java`
3. Write your names in the source files.
4. Screenshot of the unit tests passing in Eclipse with a green bar and 11/11 tests passing. You can add more tests if you want, but they must pass.
5. 1-page report with the following:
  - (a) What about the programming assignment did you like and why?
  - (b) What did you dislike and why?
  - (c) What would you change about our software (Cougar<sup>2</sup>), JUnit, EasyMock, or the test code to make your programming easier? Extra credit if you can suggest a better architecture for the software.