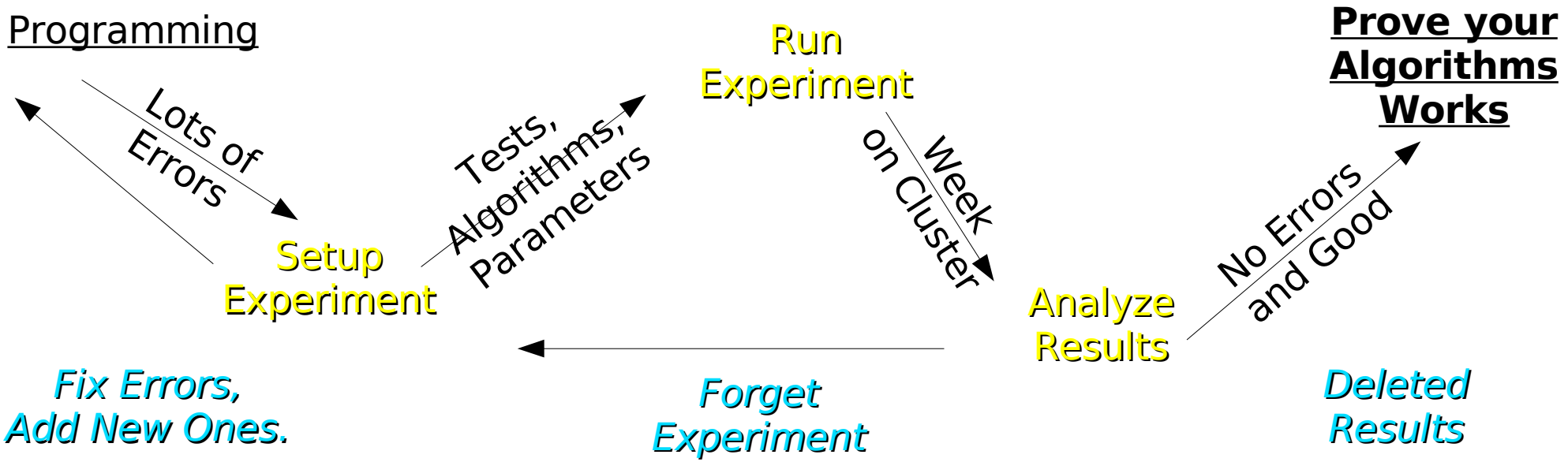
Cougar^2 Framework

Abraham Bagherjeiran
Lead Developer
University of Houston

Be the Designer



- What is wrong with this?
- What can I do about it?
- Why should I care?

Solutions

- Unit Tests
  - Code Coverage: Test Everything, All the Time
- Experiments
  - Store Parameters in Source Control
- Results
  - Store Intermediate Results to File
  - Verify Results and Reuse
- Changing Code

Introduction

- Software Framework (Our Solution)
- Design Goals
- Architecture
- Tutorial

Cougar^2 Framework

- Machine Learning and Data Mining
- Improvement to WEKA
  - Specialized Datasets: Numeric, Nominal, etc.
  - Learned Models Stored as Data
- Supports
  - Unit Testing for Learning Algorithms
  - Experiments on Cluster
  - Interface Languages: Matlab, Jython

Design: Goals

- Data-Centric Design
  - Algorithms are designed for different data.
  - Datasets organize data.
  - Attributes store data.
- Detect and diagnose errors early.
- Reuse
  - Reuse Models, Configuration
- Easy to Unit Test

Design: Evaluation

- How do you measure quality of design?
- “The design is good because I did it.”
- Object-Oriented Design Principles
- To name a few
  - High Cohesion, Low coupling
  - Open-Closed Principle
  - Liskov’s Substitution Principle
  - Dependency Inversion Principles
  - Form Follows Use
  - Design to be Tested

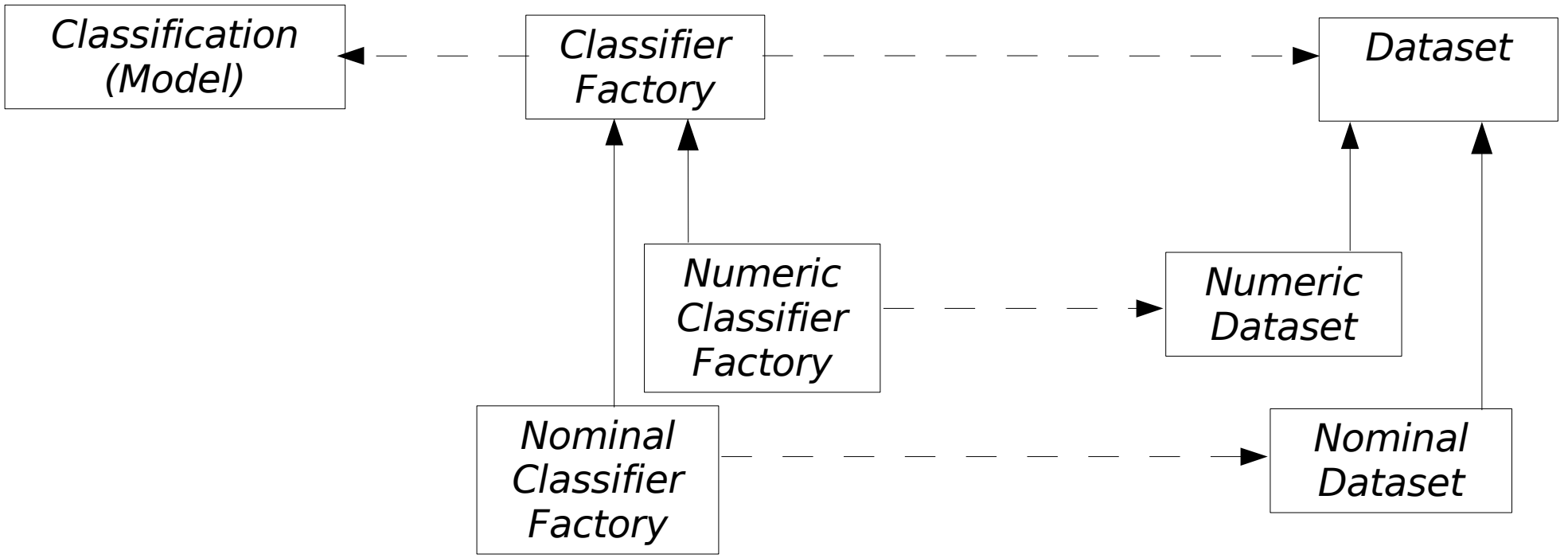
Design: Why Should I Care?

- Cannot Anticipate Changes
  - Run on Database
  - Reuse in Experiments
- Publications
  - Verify Experiments
  - Code Changes
  - People Graduate
- Doing a Good Job

Architecture

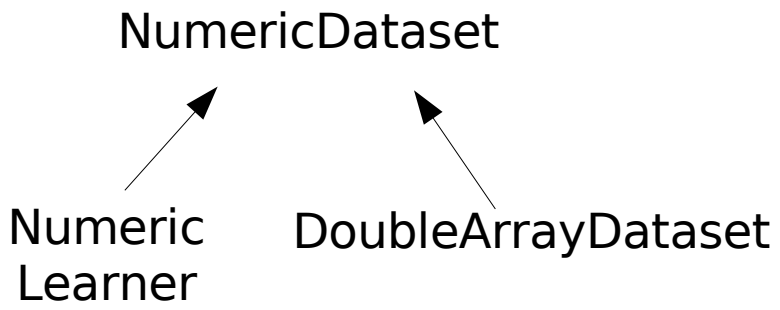
- A Dataset stores data.
- A Learner builds a Model on a Dataset.
- A Factory creates a Learner.

Architecture: Overview



Architecture: Packages

- Top-Level
  - Interfaces
  - Exceptions
  - Utility Abstract Classes
- Middle-Level
  - Default Implementations
  - Not Usually Visible
- Outside Packages
  - Access only Top-Level Interfaces

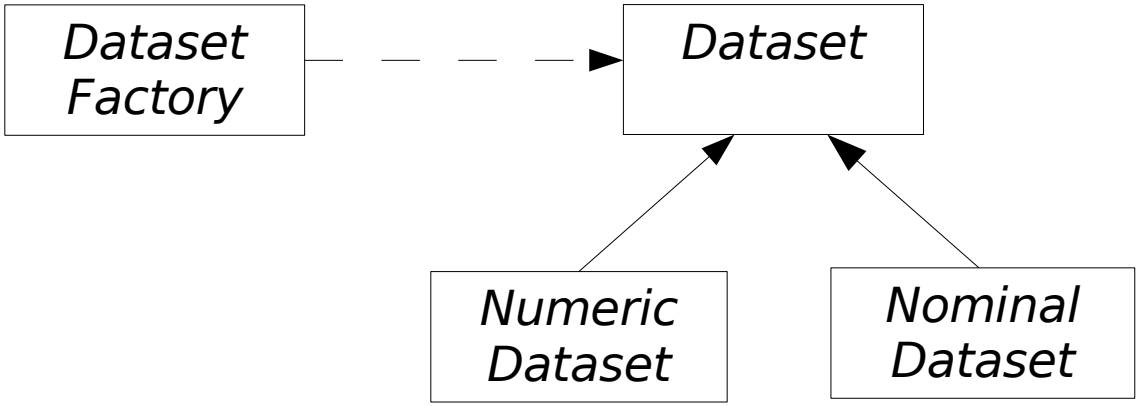


Datasets: Goals

- How to organize data?
  - Matrix, Database, Table
- What are the data?
  - Attribute: What kind of Data?
  - Data: Vector of Points
  - Meta-Data: ID, Class Label, Fitness, Cluster
- How to access?
  - get(i,j) or instance(i).value(j)

Datasets: Design

- Assumptions
  - Numeric, Nominal, Mixed
- Not Substitutable
- Fixed Structure

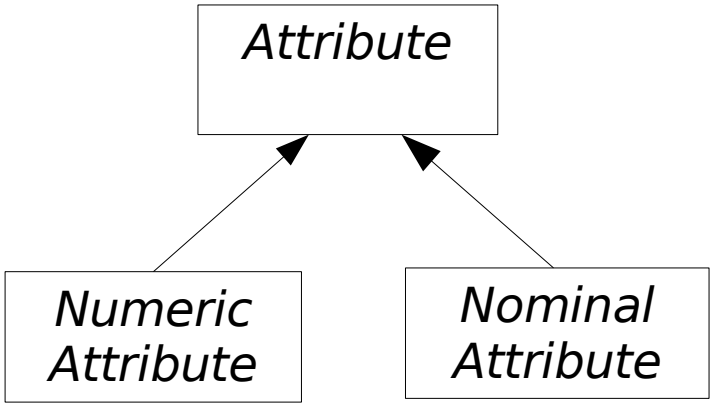


Attributes: Goals

- What are the values?
  - Integer: {0, 1, 2,…}
  - Double: [0,1.77]
  - Nominal: {“Yes”, “No”, “Maybe”}
- How are they encoded?
  - Stored as **int**, **double**
  - Range of values
  - List of Keys
- How to read from Object?

Attributes: Design

- Read from Object
- Specific Information



Attributes: Interface

```
package dmml.datasets;

public interface Attribute extends Serializable
{
  public <V> V castValue(Object o);

  public boolean isCompatible(Attribute rOther);

  public <V> V getDefault();
}

package dmml.datasets.numeric;

public interface NumericAttribute extends Attribute
{
  public boolean isInRange(double x);
  public double unitIntervalToRange(double x);
  public Double getMax();
  public Double getMin();
}
```

Store in File

Read from Object

Talk to Attributes

Default Values

Deal only with double

Datasets: Interface

```
package dmml.datasets;

public interface Dataset extends Serializable
{
  int size();
  Object getObject(int iExample, int iAtt);
  void setObject(int iExample, int iAtt, Object rValue);

  Object getMeta(int iExample, int iAtt);
  void setMeta(int iExample, int iAtt, Object rValue);

  Iterable<Integer> examples();
  Iterable<Integer> attributes();

  Attribute getAttribute(int iAtt);
  Attribute getMetaAttribute(int iAtt);

  boolean isCompatible(Dataset rOther);
  Dataset allocate();
}
```

Store in File

Data as Object

Meta-Data

Iterator over Index

Attribute Information

Talk to Datasets

Specialized Datasets: Interface

```
package dmml.datasets.numeric;
public interface NumericDataset extends Dataset
{
  public void set(int iExample, int iAtt, double rValue);
  public double get(int iExample, int iAtt);

  public NumericAttribute getAttribute(int iAtt);

  public NumericDataset allocate();
}

package dmml.datasets.nominal;
public interface NominalDataset extends Dataset
{
  public void set(int iExample, int iAtt, int rValue);
  public int get(int iExample, int iAtt);

  public NominalAttribute getAttribute(int iAtt);

  public NominalDataset allocate();
}
```

Data as double

Data are Numeric

Allocates Numeric Datasets

Data as int

Data are Nominal

Allocates Nominal Datasets

Learners

- Depend Only on Interfaces
  - Numeric Learner -> Numeric Dataset
  - Nominal Learner -> Nominal Dataset
- Check Parameters before Running
- Compiler Enforces Constraints
- Only Obtain Usable Learner

Factory

- How to Configure a Learner
  - Store Configuration
  - Resist Change
  - Small Storage
- Factory
  - Independent of Learner
  - Promotes Cohesion
  - Allows Checking

Factories

```
package dmml.classifiers;

public interface ClassifierFactory<D extends Dataset> extends Serializable
{
  public Callable<? extends Serializable>
    create(D rDataToCheck, NominalAttribute rAttToCheck)

    throws ClassifierCreationException;

}
```

- Serializable: Can be saved for future use.
- Callable: Builds *Serializable* model.
- Create: Checks configuration.
- Throws: Only get a good classifier.

Classifier

```
package java.util.concurrent;

public interface Callable<V>
{
    public V call() throws Exception;
}
```

- No Classifier Interface
- Only Callable
- Trivial Interface

Model

```
public interface Classification extends Serializable
{
  public void apply(Dataset rwTestData, NominalAttribute wPredictions);
}
```

- Model: Saved for later analysis.
- Classification: Result of learning.
- Apply: Test the model on new data.
- Store predictions to different attribute.

Code Example

```
void buildLearner(NominalClassifierFactory fac,
   NominalDataset d) {
    int x = d.get(iRow,iCol);
    fac.create(d, class).call();
}
void buildLearner(NumericClassifierFactory fac,
   NumericDataset d) {
    double x = d.get(iRow,iCol);
    fac.create(d, class).call();
}
void tryToBuild(NominalFactory nomL, NumericFactory
   numL, NominalDataset nomD, NumericDataset numD) {
    buildLearner(nomL, numD);
    buildLearner(nomL, nomD);
    buildLearner(numL, numD);
}
```

Easy to get **int** *data. Casting is not necessary.*

Can only call learner if parameters, dataset and class label are acceptable.

Compiler Error: Cannot build a Nominal Learner on a Numeric Dataset.

Tutorial

- Test-First Development
- Example
- Simple Classifier
- Mock Objects

Test-First Development

- Red-Green-Refactor
- Write the Tests First
- Must Fail Because No Code (*Red*)
- Write Code
- Must Pass (*Green*)
- Refactor

Unit Tests with JUnit 4

- Sequence
  - Setup
  - Assert Statements
  - Tear Down
- Setup
  - Create Objects for Test (Not Constructor).
- Asserts
  - Throws Error on Failure

Example

- Test division Function
  - double divide(double x, double y);
- Write Test Cases
  - Expected output for input
- Write Code
  - Minimally pass test case
- New Tests
  - More code

Test Cases / Code to Pass

- 1: 4/2 == 2

```
assertEquals(2.0, divide(4, 2));
```

- 2: 1/1 == 1

```
assertEquals(1.0, divide(1, 1));
```

- 3: 1/0 -> Exception

```
@Test(expected=Bad.class)
public void divideZero throws Bad {
    divide(1, 0);
}
```

- 1:

```
double divide(double x, double y) {
return 2;
}
```

- 2:

```
double divide(double x, double y) {
return x / y;}
```

- 3:

```
double divide(double x, double y)
   throws Bad {
double q = x / y;
if (isInfinite(q)) throw new Bad();
return q;
}
```

Problem

- Range Classifier
  - If X[0] in Range, Class = 1 Else Class = 0
- Dataset
  - Must be Numeric. Nominal Not Applicable.
- Parameters
  - Expected Range

Plan Test Cases

- Expected Behavior for Known Inputs
  - Default Range: [0,0.5]
  - Get / Set Range
  - X in range, class 1
  - X not in range, class 0
  - Model does not cheat

Example

```
public class RangeClassifierTest

{
  private RangeClassifierFactory _testFactory;
  private Callable<? extends NumericClassification> _testObject;
  private NumericClassification _testModel;

  @Before
  public void setUp() throws Exception
  {
    _testFactory = new RangeClassifierFactory();
  }

  @Test
  public void testSetup()
  {
    assertNotNull(_testFactory);
    assertNull(_testObject);
    assertNull(_testModel);
  }
   ...
}
```

Data for each test.

New Object for Each Test

Test: Verify Setup is OK

Assert Statements

Test: Factory

```
@Test
public void defaultMinMax()
{
  assertEquals(0.5, _testFactory.getMax());
  assertEquals(0.0, _testFactory.getMin());
}
```

- getMax: Default Range: [0.0, 0.5]
- Result: Compiler Error, No Method
- Solution: Add the method, Pass Test

Model Does not Cheat

- Could create dataset values
  - Hard to implement
- Mock Dataset: mNotReadClass
- Exception if model tries to read class.
- Apply Classifier
- Verify: Make sure class not read.

Mock Dataset

- 2 Attribute, 3 Values, All in Range
- Class label should always be 1.
- Read Data:
  - Read attribute 0 of each example.
  - Return 0.1 (in range) each time.
- Set Class Label:
  - Call set class label with class 1
  - For all Examples

Mock Objects

- Implements Interface
- Known Behavior -> Known Responses
- Explicit
- Verify Behavior
- EasyMock http://www.easymock.org
- Easier than implementing everything

Model Does not Cheat

```
@Test
public void modelReadsClassValues() throws Exception
{
  NumericDataset eData = createDataInRange();
  NominalAttribute tClassAtt = (NominalAttribute) eData.getMetaAttribute(1);
  NumericDataset mNotReadClass = mockDatasetErrorOnReadClass(eData);

  _testFactory.create().call().apply(mNotReadClass, tClassAtt);

  verify(mNotReadClass);
}
```

Mock Dataset: Code

```
expect(...).andReturn(2).times(2);
expect(...).andReturn(metaAtt).once();

expect(...get(and(gt(-1), lt(3)), eq(0))).andReturn(0.1).times(2);

...setMeta(and(gt(-1), lt(3)), eq(1), eq(1));
expectLastCall().times(2);

replay(mNotReadClass);
doTest(mNotReadClass);
verify(mNotReadClass);
```

Read the right values.

Set right class label

Arm the mock

Check the mock

- Expectation: **statement**, return value, times
  - **Example in {0,1,2}, attribute 0**
  - Return a value in range: 0.1.
  - Expect exactly 2 calls.

Test-First Development

- Steps
  - Write tests that fail first. (Red)
  - Write code that makes test pass. (Green)
  - Make code easier to read. (Refactor)
- Reuse Tests
- Verify After Changes
- Promotes Maintenance

Conclusion

- Framework
  - Data-Centric Design
  - Detect Errors Early
  - Separate Concerns
- Test-First Development
  - Better Modularity and Reuse
  - Easier Maintenance
- Tutorial for Homework

Thursday's Lecture

- Before Class:
  - Install Java 1.5 from http://java.sun.com
    - Not 1.4, or 1.6
  - Install Eclipse 3.2
- During Class:
  - Install subclipse
  - Checkout code
  - Answer questions about coding.

References

- Eclipse: http://www.eclipse.org
- JUnit: http://www.junit.org
- EasyMock: http://www.easymock.org