Agent-Based Modeling and Simulation

Weka and Netlogo

Dr. Alejandro Guerra-Hernández

Universidad Veracruzana
Centro de Investigación en Inteligencia Artificial
Sebastián Camacho No. 5, Xalapa, Ver., México 91000

mailto:aguerra@uv.mx
http://www.uv.mx/personal/aguerra

August 2019 - January 2020

Universidad Veracruzana

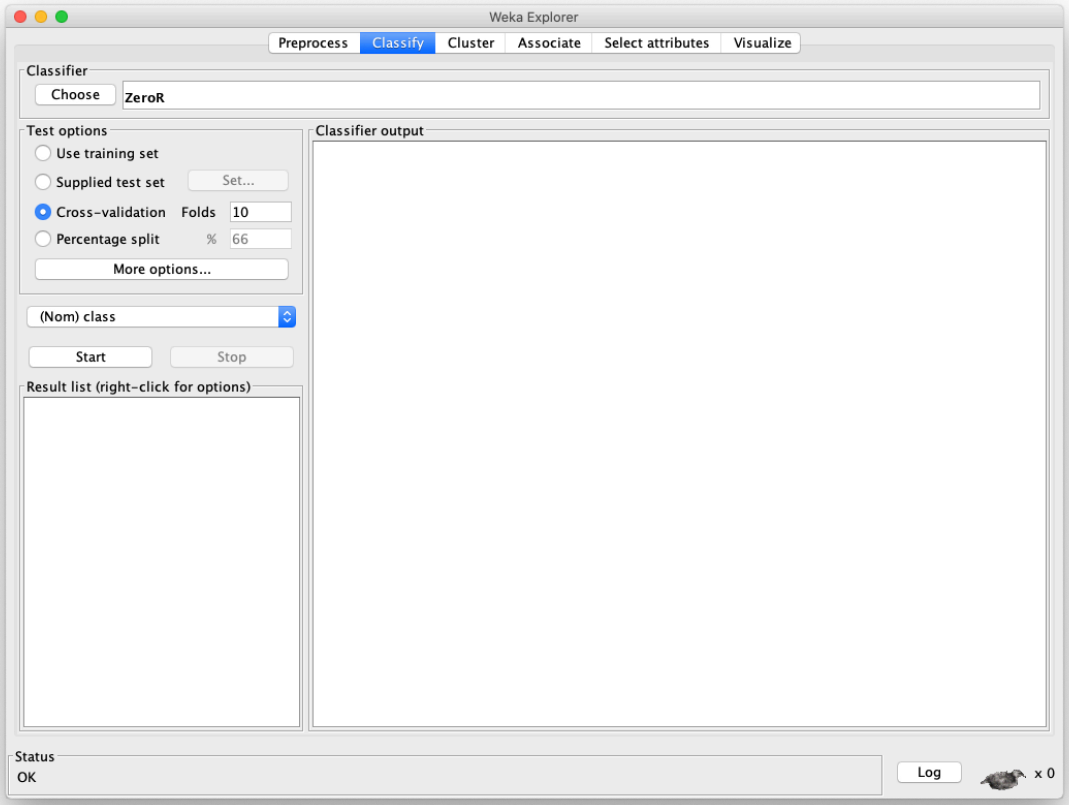
Credits

- Preliminary ideas about integrating machine learning in netlogo, all my own fault.

Universidad Veracruzana

Models as Java Objects I

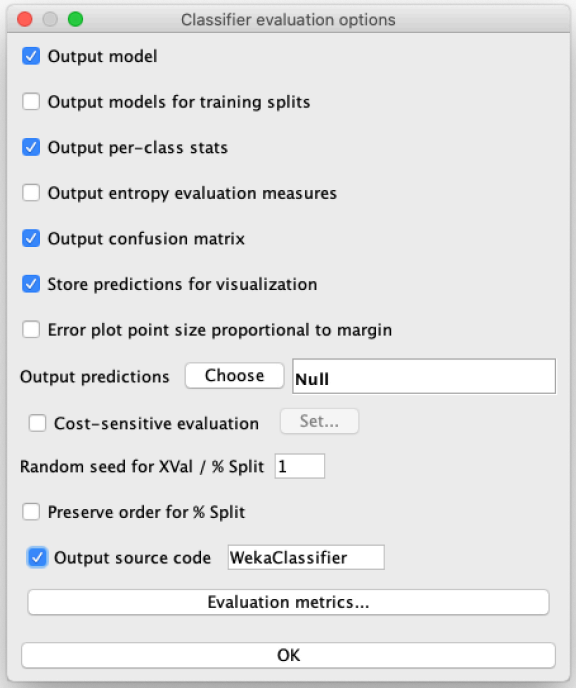
- Weka provides the option to get the classifier as source code.
- Choose the `Classify` tab to get the following screen:



Universidad Veracruzana

Models as Java Objects II

- Click on `More options...` to get the evaluation options:



- Select `Output source code` with the value `WekaClassifier`.

Universidad Veracruzana

The Decision Tree

- Load the `iris.arff` data set in Weka.
- Select `j48` as classifier.
- Run the classifier, the output includes the tree:

```
1  J48 pruned tree
2  ------------------
3
4  petalwidth <= 0.6: Iris-setosa (50.0)
5  petalwidth > 0.6
6  | petalwidth <= 1.7
7  | | petallength <= 4.9: Iris-versicolor (48.0/1.0)
8  | | petallength > 4.9
9  | | | petalwidth <= 1.5: Iris-virginica (3.0)
10 | | | petalwidth > 1.5: Iris-versicolor (3.0/1.0)
11 | petalwidth > 1.7: Iris-virginica (46.0/1.0)
```

- And (all in the Weka console) ...

Universidad Veracruzana

The Java Code for the Weka Classifier I

```
1  class WekaClassifier {
2
3    public static double classify(Object[] i)
4      throws Exception {
5
6      double p = Double.NaN;
7      p = WekaClassifier.N6e6e445a0(i);
8      return p;
9    }
10   static double N6e6e445a0(Object []i) {
11     double p = Double.NaN;
12     if (i[3] == null) {
13       p = 0;
14     } else if (((Double) i[3]).doubleValue() <= 0.6) {
15       p = 0;
16     } else if (((Double) i[3]).doubleValue() > 0.6) {
17     p = WekaClassifier.N2ad233071(i);
18     }
19     return p;
20   }
```

Universidad Veracruzana

The Java Code for the Weka Classifier II

```
21  static double N2ad233071(Object []i) {
22    double p = Double.NaN;
23    if (i[3] == null) {
24      p = 1;
25    } else if (((Double) i[3]).doubleValue() <= 1.7) {
26    p = WekaClassifier.N164d18892(i);
27    } else if (((Double) i[3]).doubleValue() > 1.7) {
28      p = 2;
29    }
30    return p;
31  }
32  static double N164d18892(Object []i) {
33    double p = Double.NaN;
34    if (i[2] == null) {
35      p = 1;
36    } else if (((Double) i[2]).doubleValue() <= 4.9) {
37      p = 1;
38    } else if (((Double) i[2]).doubleValue() > 4.9) {
39    p = WekaClassifier.N1ed25c43(i);
40    }
41    return p;
```

Universidad Veracruzana

The Java Code for the Weka Classifier III

```
42    }
43    static double N1ed25c43(Object []i) {
44      double p = Double.NaN;
45      if (i[3] == null) {
46        p = 2;
47      } else if (((Double) i[3]).doubleValue() <= 1.5) {
48        p = 2;
49      } else if (((Double) i[3]).doubleValue() > 1.5) {
50        p = 1;
51      }
52      return p;
53    }
54  }
```

Universidad Veracruzana

Observations

- This is a low leve class used by a wrapper defined next.
- Observe that the parameter of `classify` is an array of `Objects`.
- Observe the use of `NaN` to initialize variables.
- Observe the calls in cascade implementing the tests in the decision tree.
- Observe that `classify` returns a `double`, the index of the class value.
- Observe the necessary castings to `Double`.

Universidad Veracruzana

The Weka Wrapper I

```
1  // Generated with Weka 3.9.3
2  // This code is public domain and comes with no warranty.
3  // Timestamp: Sun Nov 24 19:18:19 CST 2019
4
5  package weka.classifiers;
6
7  import weka.core.Attribute;
8  import weka.core.Capabilities;
9  import weka.core.Capabilities.Capability;
10 import weka.core.Instance;
11 import weka.core.Instances;
12 import weka.core.RevisionUtils;
13 import weka.classifiers.Classifier;
14 import weka.classifiers.AbstractClassifier;
15
16 public class WekaWrapper
17   extends AbstractClassifier {
18
19   /**
20    * Returns only the toString() method.
```

The Weka Wrapper II

```
21      * @return a string describing the classifier
22      */
23
24     public String globalInfo() {
25       return toString();
26     }
27
28     /**
29      * Returns the capabilities of this classifier.
30      * @return the capabilities
31      */
32
33     public Capabilities getCapabilities() {
34       weka.core.Capabilities result = new weka.core.Capabilities(this);
35
36       result.enable(weka.core.Capabilities.Capability.NOMINAL_ATTRIBUTES);
37       result.enable(weka.core.Capabilities.Capability.NUMERIC_ATTRIBUTES);
38       result.enable(weka.core.Capabilities.Capability.DATE_ATTRIBUTES);
39       result.enable(weka.core.Capabilities.Capability.MISSING_VALUES);
40       result.enable(weka.core.Capabilities.Capability.NOMINAL_CLASS);
41       result.enable(weka.core.Capabilities.Capability.MISSING_CLASS_VALUES);
```

The Weka Wrapper III

```
42
43
44      result.setMinimumNumberInstances(0);
45
46      return result;
47    }
48
49    /**
50     * only checks the data against its capabilities.
51     * @param i the training data
52     */
53
54    public void buildClassifier(Instances i) throws Exception {
55      getCapabilities().testWithFail(i); // can classifier handle the data?
56    }
57
58    /**
59     * Classifies the given instance.
60     * @param i the instance to classify
61     * @return the classification result
62     */
```

Universidad Veracruzana

The Weka Wrapper IV

```
63
64  public double classifyInstance(Instance i) throws Exception {
65    Object[] s = new Object[i.numAttributes()];
66
67    for (int j = 0; j < s.length; j++) {
68      if (!i.isMissing(j)) {
69        if (i.attribute(j).isNominal())
70          s[j] = new String(i.stringValue(j));
71        else if (i.attribute(j).isNumeric())
72          s[j] = new Double(i.value(j));
73      }
74    }
75
76    s[i.classIndex()] = null; // set class value to missing
77    return WekaClassifier.classify(s);
78  }
79
80  /**
81   * Returns the revision string.
82   * @return the revision
83   */
```

Universidad Veracruzana

The Weka Wrapper V

```
84
85    public String getRevision() {
86      return RevisionUtils.extract("1.0");
87    }
88
89    /**
90     * Returns only the classnames and what classifier it is based on.
91     * @return a short description
92     */
93
94    public String toString() {
95      return "Auto-generated classifier wrapper, based on
96      weka.classifiers.trees.J48 (generated with Weka 3.9.3).\n" +
97      this.getClass().getName() + "/WekaClassifier";
98    }
99
100   /**
101    * Runs the classfier from command line.
102    * @param args the commandline arguments
103    */
104
```

Universidad Veracruzana

The Weka Wrapper VI

```
105     public static void main(String args[]) {
106       runClassifier(new WekaWrapper(), args);
107     }
108   }
```

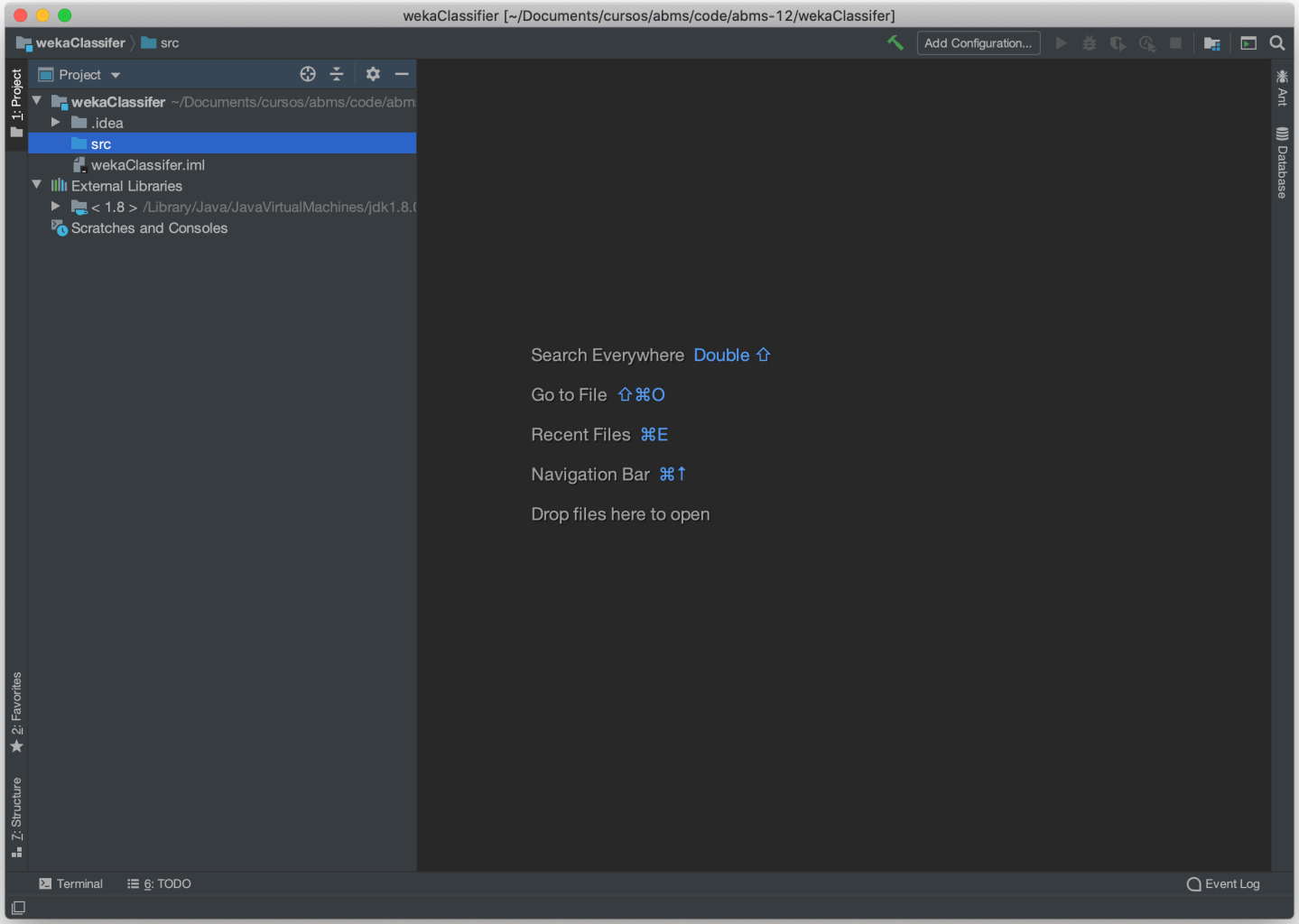
Universidad Veracruzana

Creating a Project with the Code

- I'll use IntelliJ IDEA as IDE, since we will use it later to code the NetLogo extension based on this classifier generated with Weka.
- For the same reasons use Java version 1.8 (the JVM used by NetLogo).
- Create a new Java project and call it `wekaClassifier`.
- If everything goes fine, you have an empty project as shown in the next slide.

Universidad Veracruzana

IntelliJ wekaClassifier Project



ersidad Veracruzana

Compiling the Code

- Add a source file called `WekaWrapper` and copy there the code generated by Weka.
- A lot of errors are detected since our project knows nothing about Weka! Add `weka.jar` to the external libraries of the project.
- Select `File` and then `Project Structure`. Then `Libraries` and add the `weka.jar` that is in the distribution folder of Weka.
- Compile the project, an `out` directory is created in your project. It includes a `production/wekaClassifier` sub-folder. There you will find the classes compiled.

Universidad Veracruzana

Testing the Wrapper I

- Add a file called `TestClassifier` with the following content:

```
1  import java.io.*;
2
3  import weka.core.Instances;
4
5  public class TestClassifier {
6      public static void main(String[] args) {
7          WekaWrapper ww = new WekaWrapper();
8          try {
9              Instances unlabeled = new Instances(
10                     new BufferedReader(new FileReader("iris-test.arff")));
11             unlabeled.setClassIndex(unlabeled.numAttributes() - 1);
12             for (int i = 0; i < unlabeled.numInstances(); i++) {
13                 double clsLabel = ww.classifyInstance(unlabeled.instance(i));
14                 System.out.println(clsLabel + " -> " +
15                         unlabeled.classAttribute().value((int) clsLabel));
16             }
17         } catch (FileNotFoundException e) {
18             e.printStackTrace();
19         } catch (IOException e) {
20             e.printStackTrace();
21         } catch (Exception e) {
22             e.printStackTrace();
23         }
24     }
25 }
```

Universidad Veracruzana

Testing the Wrapper II

- Compile the project, now you have four classes in the `out` folder.
- Add a file `iris-test.arff` in the directory where you have the classes with the following content:

```
1  @RELATION iris
2
3  @ATTRIBUTE sepallength REAL
4  @ATTRIBUTE sepalwidth  REAL
5  @ATTRIBUTE petallength  REAL
6  @ATTRIBUTE petalwidth REAL
7  @ATTRIBUTE class  {Iris-setosa,Iris-versicolor,Iris-virginica}
8
9  @DATA
10 5.1,3.5,1.4,0.2,?
11 4.9,3.0,1.4,0.2,?
12 7.0,3.2,4.7,1.4,?
13 6.4,3.2,4.5,1.5,?
14 6.3,3.3,6.0,2.5,?
15 5.8,2.7,5.1,1.9,?
```

- Now you can run the test:

Universidad Veracruzana

Testing the Wrapper III

```
> java -classpath .:/Applications/weka-3-9-3/weka.jar TestClassifier
0.0 -> Iris-setosa
0.0 -> Iris-setosa
1.0 -> Iris-versicolor
1.0 -> Iris-versicolor
2.0 -> Iris-virginica
2.0 -> Iris-virginica
```

Universidad Veracruzana

Implementing Classify I

- What I really need is a way of classifying a string representing an unknown instance, e.g., `"5.1,3.5,1.4,0.2"`.
- A first attempt is implementing a `Classify` class that works as follows:

```
1 > java -classpath .:/Applications/weka-3-9-3/weka.jar Classify "5.1,3.5,1.4,0.2"
2 5.1,3.5,1.4,0.2,?
3 Iris-setosa
```

- Add a file `Classify.java` to the project with the following code:

Universidad Veracruzana

Implementing Classify II

```
1  import weka.core.Attribute;
2  import weka.core.DenseInstance;
3  import weka.core.Instance;
4  import weka.core.Instances;
5
6  import java.io.*;
7
8  public class Classify {
9      public static void main(String[] args) throws IOException {
10         String arg = args[0];
11         WekaWrapper ww = new WekaWrapper();
12         Instances testSet = new Instances(
13                 new BufferedReader(new FileReader("iris-test.arff")));
14         int numAttributes = testSet.numAttributes();
15         Instance inst = new DenseInstance(numAttributes);
16         inst.setDataset(testSet);
17         testSet.setClassIndex(numAttributes - 1);
18         String[] vals = arg.split(",");
19         Attribute[] attributes = new Attribute[numAttributes];
20         for(int i=0; i<numAttributes-1; i++) {
21             attributes[i] = testSet.attribute(i);
22         }
23
24         int j = 0;
25         for(String val:vals){
26             if(attributes[j].isNumeric()){
27                 double valDouble = Double.parseDouble(val);
28                 inst.setValue(attributes[j],valDouble);
29             } else {
```

Universidad Veracruzana

Implementing Classify III

```
30                  inst.setValue(attributes[j], val);
31              }
32              j++;
33          }
34          try {
35              System.out.println(inst);
36              double clsLabel = ww.classifyInstance(inst);
37              System.out.println(testSet.classAttribute().value((int) clsLabel));
38          } catch (Exception e) {
39              // TODO Auto-generated catch block
40              e.printStackTrace();
41          }
42      }
43  }
```

- Recompile the project and run classify as above.

Universidad Veracruzana

Preliminares

- NetLogo is implemented in Scala, so you need to install it. In Mac OS: `brew install scala`. I have installed version 2.13.1
- You also need Java version 1.8 because NetLogo use the JVM of that particular version.
- We will use IntelliJ the Java IDE by JetBrains©. The choice is due to its capability to work also with Scala.
- We will create a new class for our classifier, lets called it `Classify`.

Universidad Veracruzana

External Libraries

- Add the module for Scala in IntelliJ, the path to it is: `/user/local/opt/scala/idea`
- Add NetLogo as an external library in IntelliJ, the path to it is: `/Applications/Netlogo 6.1.1/Java/netlogo6.1.1.jar`
- Add SamIam as an external library too, its path is: `/Applications/samiam/inflib.jar`
- Of course, the exact paths depend on your own installation.

Universidad Veracruzana

The Manifest

- Our project must include a file called `manifest.txt` which content is as follows:

```
Manifest-Version: 1.0
Extension-Name: coopBN
Class-Manager: CoopBNExtension
NetLogo-Extension-API-Version: 6.1.1
```

- The extension name will be used to load the extension in NetLogo.
- The class manager will be implemented next.

Universidad Veracruzana

The Class Manager CoopBNExtension

```
1 import org.nlogo.api.*;
2
3 public class CoopBNExtension extends DefaultClassManager {
4
5     public void load(PrimitiveManager primitiveManager) {
6         additionalJars().add("inflib");
7         primitiveManager.addPrimitive("get-prob", new CoopBN());
8     }
9 }
```

Universidad Veracruzana

Observations

- Although `inflib.jar` has been added as an external library to our project, it is necessary to register it as an additional Jar in the class manager.
- We declare then a new primitive called `get-prob` as an instance of the class `CoopBN` defined next.
- Primitives can be reports or actions.
- Our report will be called as `coopBN:get-prob` in NetLogo.

Universidad Veracruzana

Referencias I

Universidad Veracruzana