Notes on Using WEKA for Implementing a Custom Classifier.

This document assumes some familiarity with Java or some other Object Oriented Language (C++, C#, etc)

Introduction

WEKA is a collection of Java classes (class framework) that one needs to use to implement a classifier that can take advantage of functionality already accumulated in WEKA. The degree to which one would want to take advantage of WEKA functionality can be different but there is a bare minimum discussed in this document.

What WEKA Provides

WEKA Explorer is an application that provides the following functionality:

- Dataset Management: loading data, feeding them to classifiers, filters, storing the results of classification, apportioning data between training and testing subsets.
- Preprocessing of data using wide range of preprocessing filters. WEKA Explorer allows selection of a filter
- Classification, WEKA Explorer allows selection of a classifier
- Reporting classification results including visualization.

How WEKA Can Be Extended

One can extend WEKA by providing a custom classifier or a custom filter or many other things beyond the scope of this document.

What Needs to Be Done To Add a Custom Classifier To WEKA

In order for your classifier to appear in the menu and be callable by WEKA Explorer, one needs to implement a custom “classifier” class.

You need to extend provided by WEKA “Classifier” class.

At the minimum you need to implement just 3 methods.

1. BuildClassifier
2. ClassifyInstance
3. getOptions
**buildClassifier Method**

```java
public void buildClassifier(Instances data)
```

This method will be called by WEKA Explorer when you select a classifier. It will receive a training dataset as a parameter. You can do whatever you want with the data it is totally up to the implementer but somehow you should build and store all the information necessary for classification. Look at the appendix 1 to see how to access individual data to store them for simplified NN classifier.

**classifyInstance**

```java
public void classifyInstance(Instance instance)
```

This method will be called by WEKA Explorer after your training is complete. It will receive a single instance as a parameter. You need to classify it (no restrictions) and return a class value you decided for this sample of data. Look at the appendix to see how to classify an instance using simple NN classifier.

**Other WEKA Methods You Need to Know**

```java
int Instances.numAttributes(); //returns # of Attributes
int Instances.numInstances(); //returns # of samples
int Instances.numClasses(); //returns # of classes
Enumeration Instances.enumerateInstances(); //enumerate instances
double Instance.value(int i); //returns value of particular attribut
```

**Appendix**

```java
public class NN extends Classifier {
    Instances m_Instances;
    int m_nAttributes;
    //this method build your model
    public void buildClassifier(Instances data) {
        m_Instances = new Instances(data);
        m_nAttributes = data.numAttributes();

        Enumeration enu = m_Instances.enumerateInstances();
        while (enu.hasMoreElements()) {
            Instance _instance = (Instance) enu.nextElement();
            int n = _instance.numAttributes();
            double x,y;
            for (int i = 0; i < n; i++) {
```
x = _instance.value(i);
y = x;
}
}
//this method lets you to test the instance using the classifier
public double classifyInstance(Instance instance)
{
    Enumeration enu = m_Instances.enumerateInstances();
    double distance = 9999999;
    double classValue = -1;
    while (enu.hasMoreElements()) {
        Instance _instance = (Instance) enu.nextElement();
        double _distance = CalculateDistance(instance,_instance);
        if (_distance < distance)
        {
            distance = _distance;
            classValue = _instance.classValue();
        }
    }
    return classValue;
}
public double CalculateDistance(Instance i1, Instance i2)
{
    double s=0;
    for (int i = 0;i < m_nAttributes-1;i++)
    {
        double p = (i1.value(i)-i2.value(i));
        s += p*p;
    }
    return s;
}
//this function is used to show the list of the parameter options
public String [] getOptions() {
    String [] options = new String [2];
    while (current < options.length)
    {
        options[current++] = " ";
    }
    return options;
}