Dartmouth CS 65/165
Smartphone Programming **Professor Andrew T. Campbell**

MyRuns Data Collector and Weka Tutorial

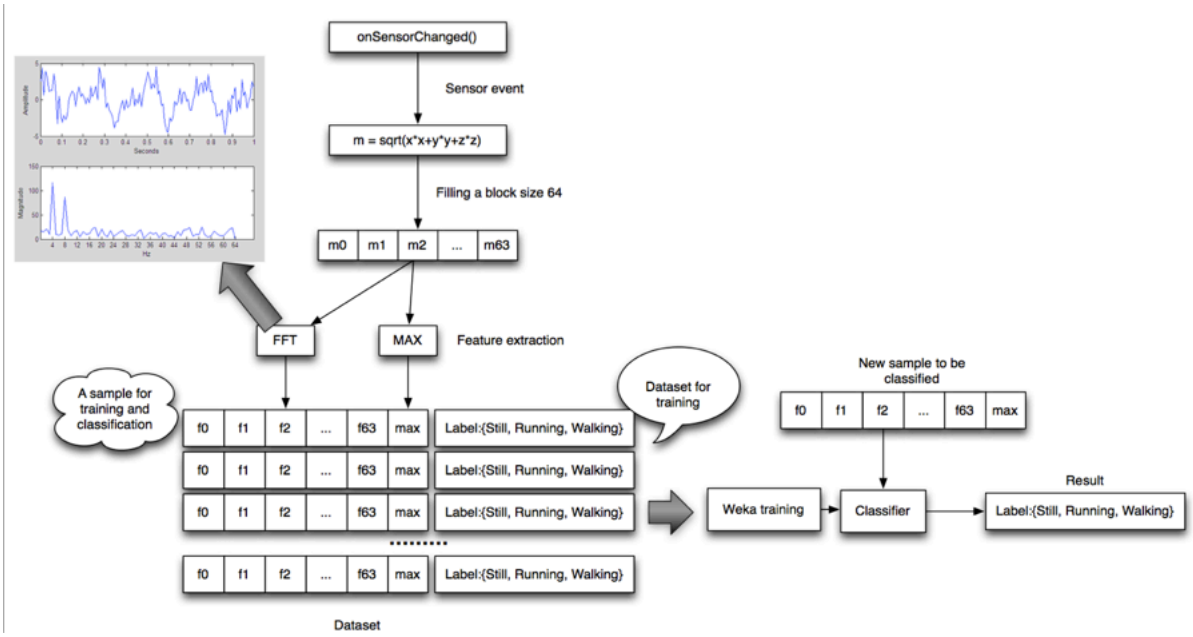
In this note, I will show how to collect activity data using MyRuns Data Collector and teach you how to play with Weka GUI to train your own classifier.

Training and Classification workflow

The diagram below shows both the training phase and classification phase. Let's discuss this diagram in a little more detail so you have a good sense of what needs to be computed in both phases. Note, there is a lot of code reuse between coding the collector (which used for training phase) and the classification phase which supports by MyRuns automatic mode.

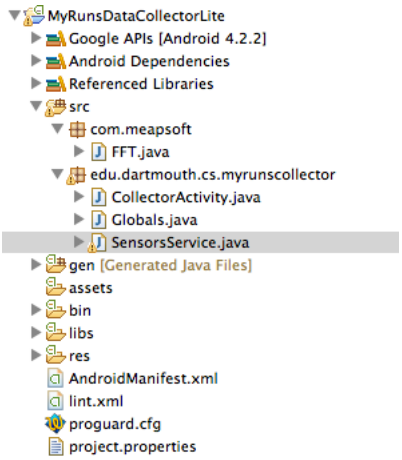
In the training phase a method called onSensorChanged() produces sensor samples (x, y, z) in a time series (each time onSensorChanged() is called), which are used to compute m (magnitude) of the sensor samples. The workflow buffers up 64 consecutive magnitudes (m0..m63) before computing the FFT for each of the magnitudes (f0..f63) – each of these features is called an FFT coefficient; e.g., f0 and f63 are the low and highest frequency coefficients.

As shown the upper left illustration in the diagram, FFT transforms a time series of amplitude over time to magnitude (some representation of amplitude) across frequency; the example shows some oscillating system where the dominate frequency is between 4-8 cycles/second – imagine a ball attached to an elastic band that this stretched and oscillates for a short period of time. The rate of the amplitude transposed to the frequency domain may look something like the insert in the figure. The training phase also stores the maximum (MAX) magnitude of the (m0..m63) and the label the user provided to the collector (see collector UI later). The individual features that are computed magnitudes (f0..f63) and MAX magnitude in addition to the label that the user supplies. Collectively, we call these features the feature vector comprises: magnitudes (f0..f63), MAX magnitude, label. Because the user can collect a small or larger amount of data, there may be a small or large number of vectors – say the differences between running for 1-10 minutes. The longer you collect data the more feature vectors accumulated. Once the user has stopped collecting training data using the collector tool the weka.jar APIs are used to format the feature vectors in the correct format to upload to your laptop—we call this the training data set (the file name is called features.arff). To complete the training phase we run Weka GUI to generate the classifier. The creation of the classifier is the completion of the training phase. In the classification phase, MyRuns uses exactly the same workflow of the training phase in terms of computing the feature vectors (sampling x,y,z, computing the magnitudes – and MAX – and FFT coefficients) but this time there is no user supplied label as in the collector. The live systems classification phase does not need the user to supply a label because it has trained a model (called classifier) that can infer the label (e.g., walking) from the feature vectors. The sub-pipeline of computing the feature vectors is simply shown in the diagram as "new samples to be classified" that are then fed into the classifier – the result is a label. This provides a good overview of what your collector code and extensions for MyRuns for the automatic mode need to do.



MyRuns Data Collector

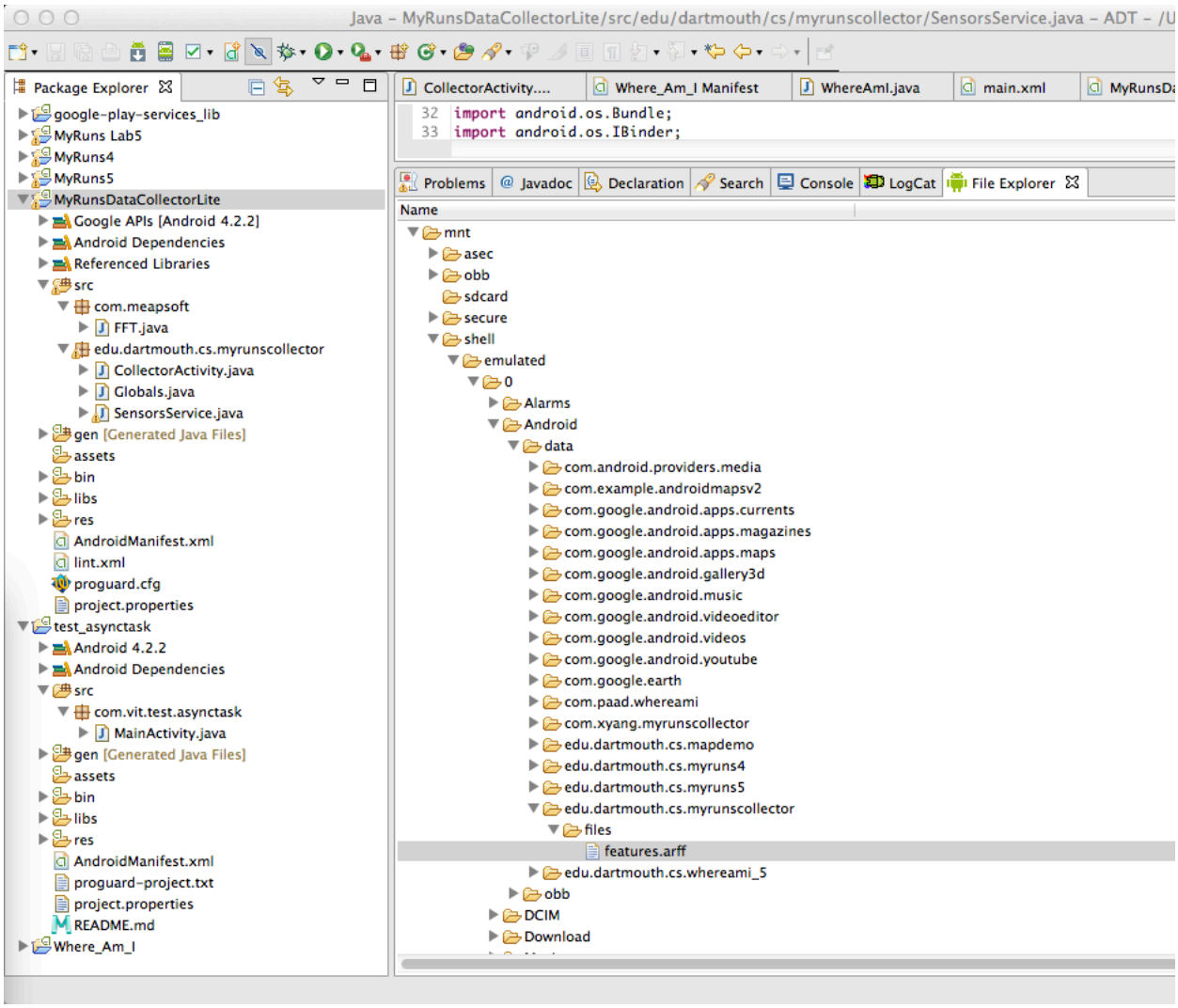
We now discuss the project design for the collector. As shown below the collector consists of the CollectorActivity, Globals (data) and SensorService. The project also shows the FFT.java and the Weka.jar.



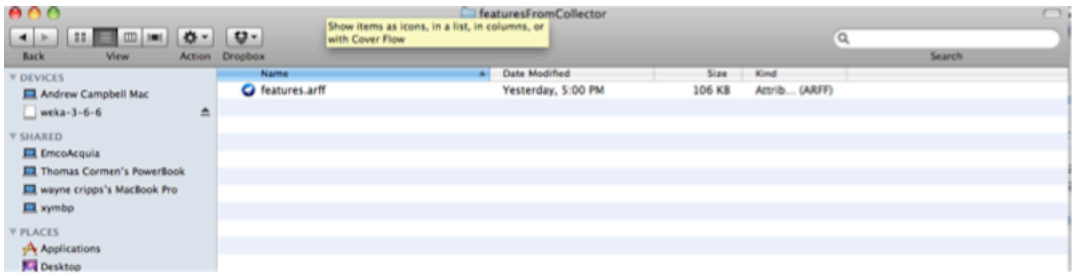
The goal of this tool collect sensor data for the four different classes (standing, walking, running, others). The user selects an activity (thereby providing the training phase the label it needs to associate with the data it's about to collect – this is called supervised learning in Machine Learning parlance). Once an activity is selected (e.g., walking) the user clicks "start collecting" and starts to walk around for at least 3 minutes. Note, that once the user selects start collecting "Delete data" button (discussed below) is disabled – you don't want to accidentally delete the data. The user repeats this for each of the activities --- pressing stop at the end of the activity collection phase and then selecting a new activity and starting to collect again – so, yes, in the case of running you need to take a 3 minute run (tip: while debugging you don't need to run during code updates – simple emulate standing, walking and running by shaking the phone in your hand; once you have solid code then go for the real run, etc.).

The delete button removes the data, which is stored in a file on the phone (called features.arff). If you want to start all over hit delete before starting else the new collected data will be appended to the end of the file. You could collect multiple instances of training data – go for multiple runs, walks and different times and it should all be accumulated in the same file. The collector implements the training phase shown in the diagram and discussed in the section above. The result of the collector phase is the creation of a Weka formatted features.arff file. So the collector needs to be able to collect accelerometer samples, compute the magnitudes and use the FFT.java class to compute the coefficients, etc. to produce the feature vectors. The weka.jar APIs are used to format the feature vectors into to the correct format for the Weka tool that produces the classifier, as discussed in the next section.

Next, we need to upload the saved data to laptop. File Explorer in your Eclipse helps to do that.



You need to highlight the file (features.arff) and click upload – red arrow points to it in the DDMS view.

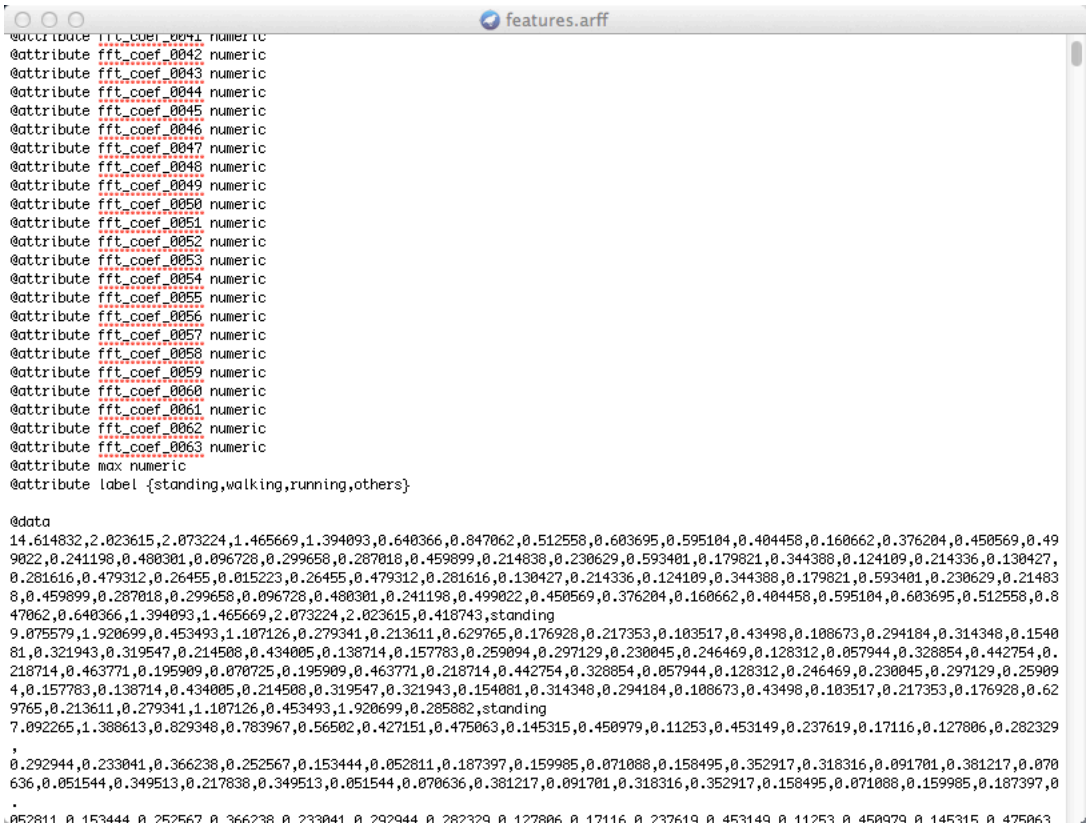


Weka Tutorial

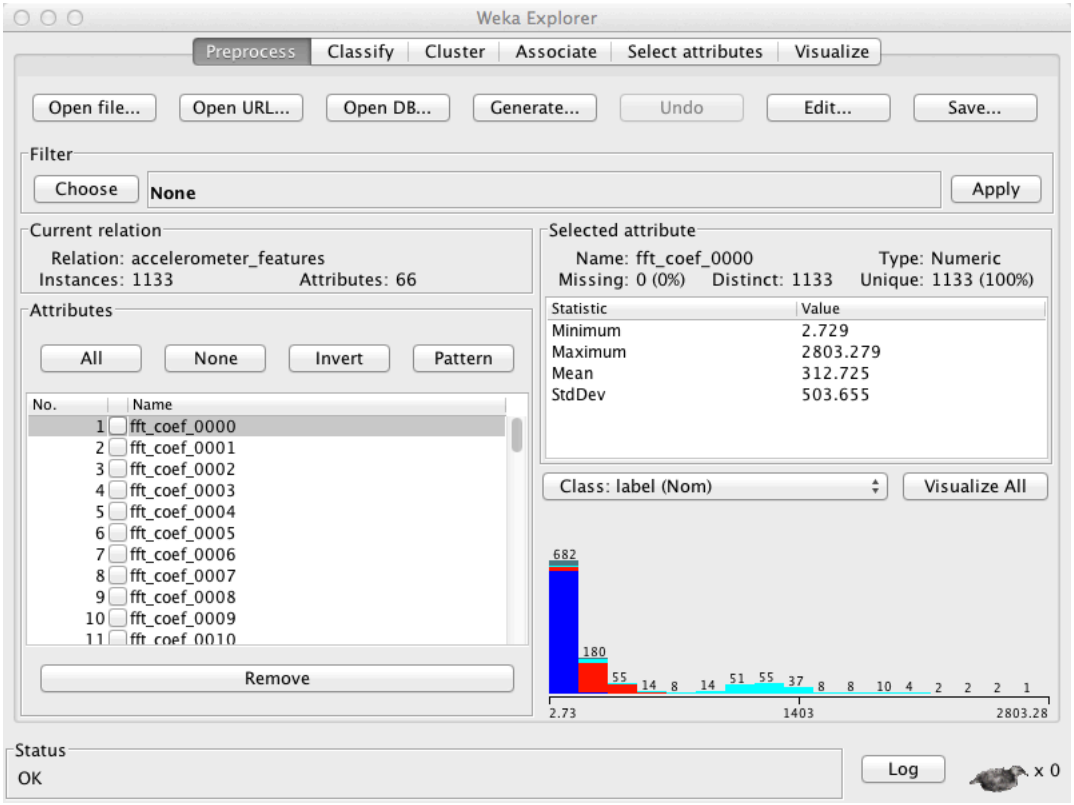
This section guides you to setup the Weka GUI and shows how to generate the java implementation of a decision tree.

Installing Weka: Once the file is uploaded, my mac knows it is a Weka file as shown with the Weka bird icon on the file. Clearly you need to have downloaded and installed Weka for the system to recognize the file extension. Therefore, you need to install Weka:

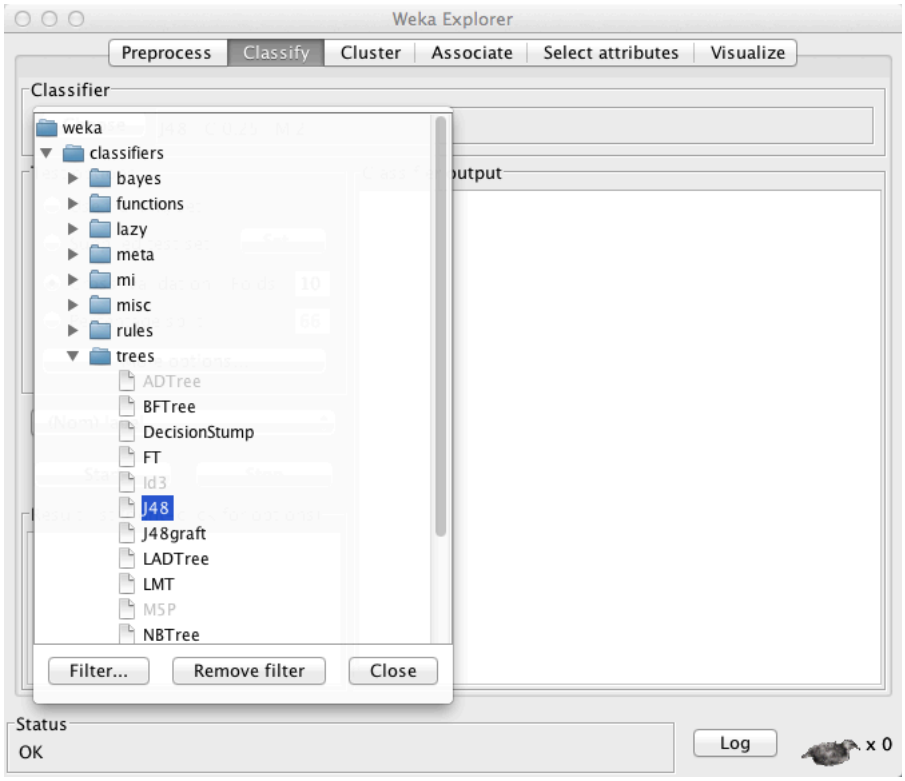
Once Weka is installed and the features.arff file uploaded it is time to get Weka to generate the classifier. Here is how you do it. First, if you open features.arff using your favorite text editor you will see the Weka format of the file. It makes complete sense. The first part of the file is the specification followed by the feature vector as shown figure below.



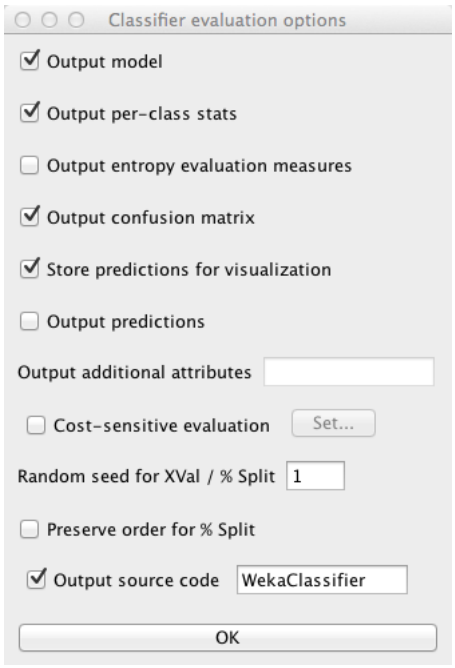
If you click on the file name the Weka fires up with the following window as shown below.



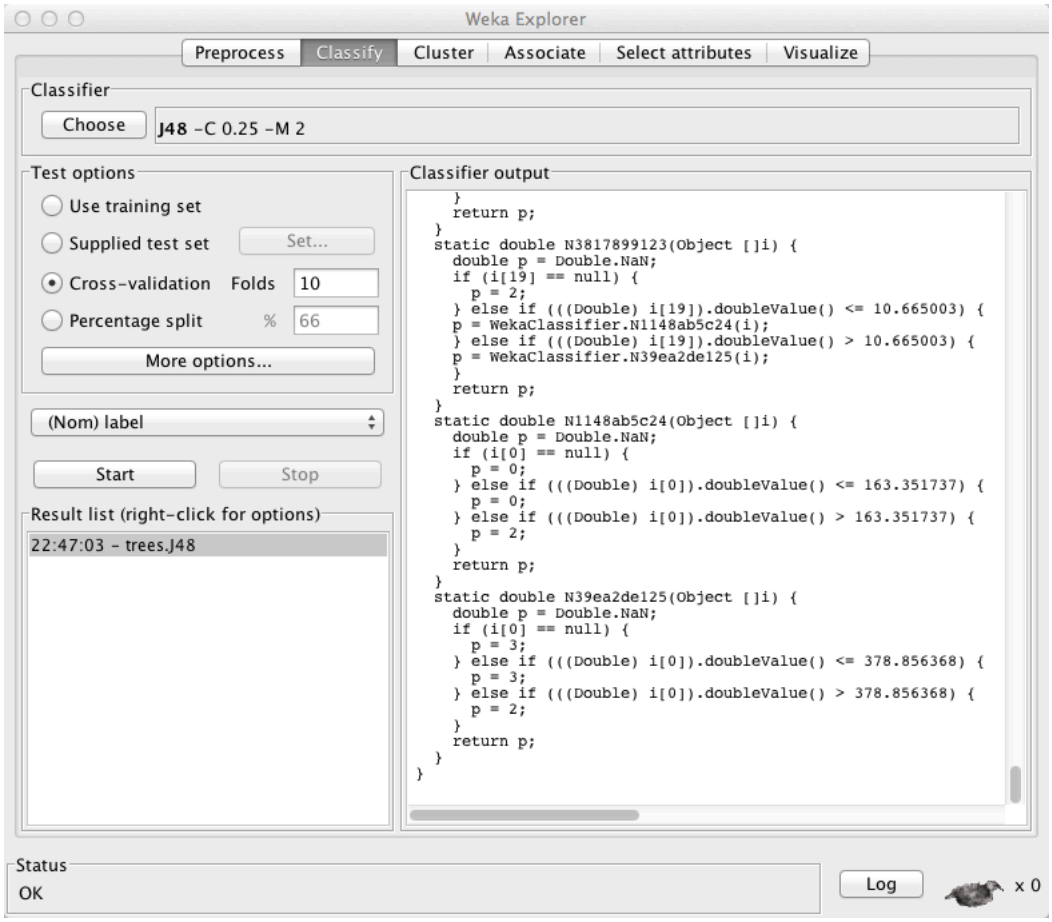
Select the Classify option – and follow the example below where the J48 decision tree is selected and then select more options to specify the source code name of the classifier object.



We name the classifier object source code – output source code WekaClassifier (that gives it the name WekaClassifier.java) as shown in the figure below.



Now we are ready to generate the classifier – and its java implementation. Simply click "Start" to generate the code and lots of other interesting information. In essence Weka takes the training data you supply to train a simple decision tree classifier.



The code produced from your training code would be different and to be honest you will not make much sense of this somewhat simple (but at the same time complex) piece of code – it is essentially a bunch of if-then statements that implement the decision tree we discussed in class.

The classifier output also gives use come very useful performance information about our classifier. First, it provides a confusion metric. This indicates when the classifier confused one class over another; for example, the confusion metric tells us that standing (i.e., still) gets confused for walking in 4 cases and running in 1 case. Similarly, the classifier is cross validated and is shown to be 91% accurate, which is quite good for our app. While we are not a Machine Learning class these are

important parts of classification performance. I am glad we expose you not only training a simple classifier but reasoning about the performance of a classifier that we trained!