Weka Lab 6

Introduction

In previous labs we have used the Weka graphical user interface to load data, create models and make predictions. In this lab we will explore two other ways to use Weka, on the command line, and from a java application. We will start by connecting to one of the EECS cycle servers, preparing and configuring the environment for weka. A tutorial is provided to demonstrate use of weka using only the command line, as well as through java code.

Connecting to the EECS cycle servers.

For this lab we will login to one of the EECS cycle servers. There are several to choose from:

cycle1.eecs.ku.edu
cycle2.eecs.ku.edu
cycle3.eecs.ku.edu

To connect, we will use a program called “PuTTy”. Once PuTTy has been started, enter the name of the server you wish to connect to, select SSH as the connection type, use port 22. This is shown in Illustration 1. Once connected, you will be prompted for your EECS username, this is the same username that you use to login to the windows machines in the lab.

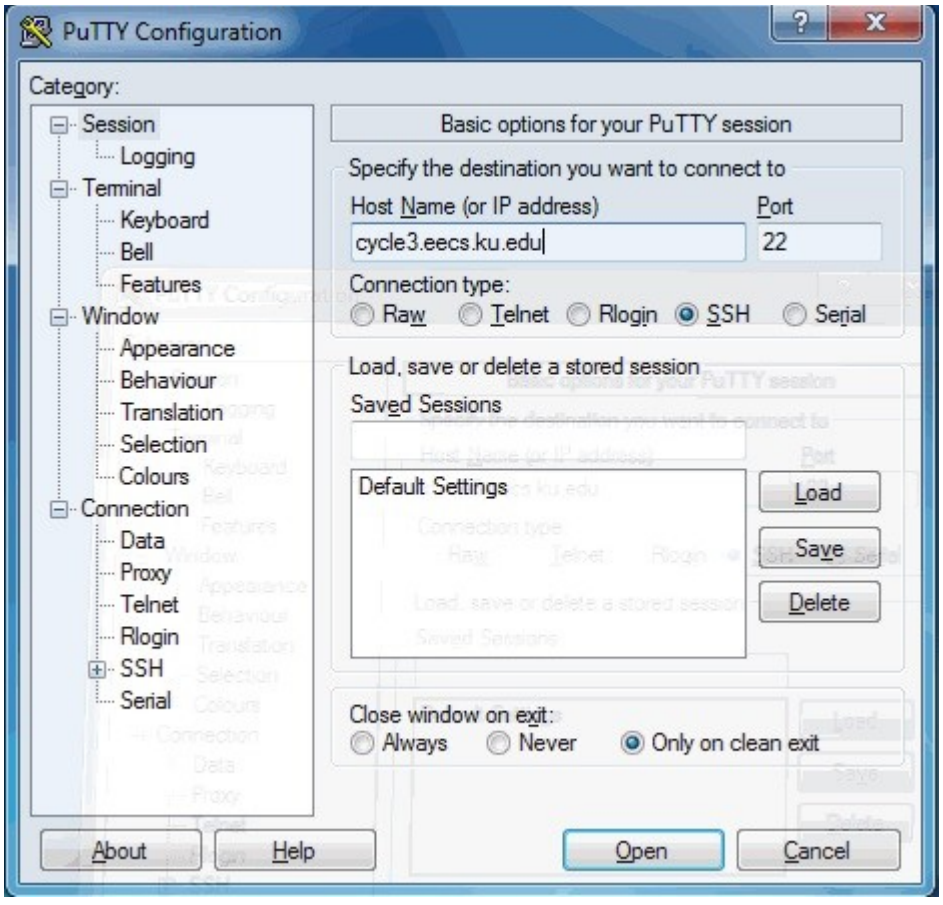

Illustration 1: Settings to connect to EECS server cycle3, using SSH on port 22. Note, you may also use cycle1 or cycle2.

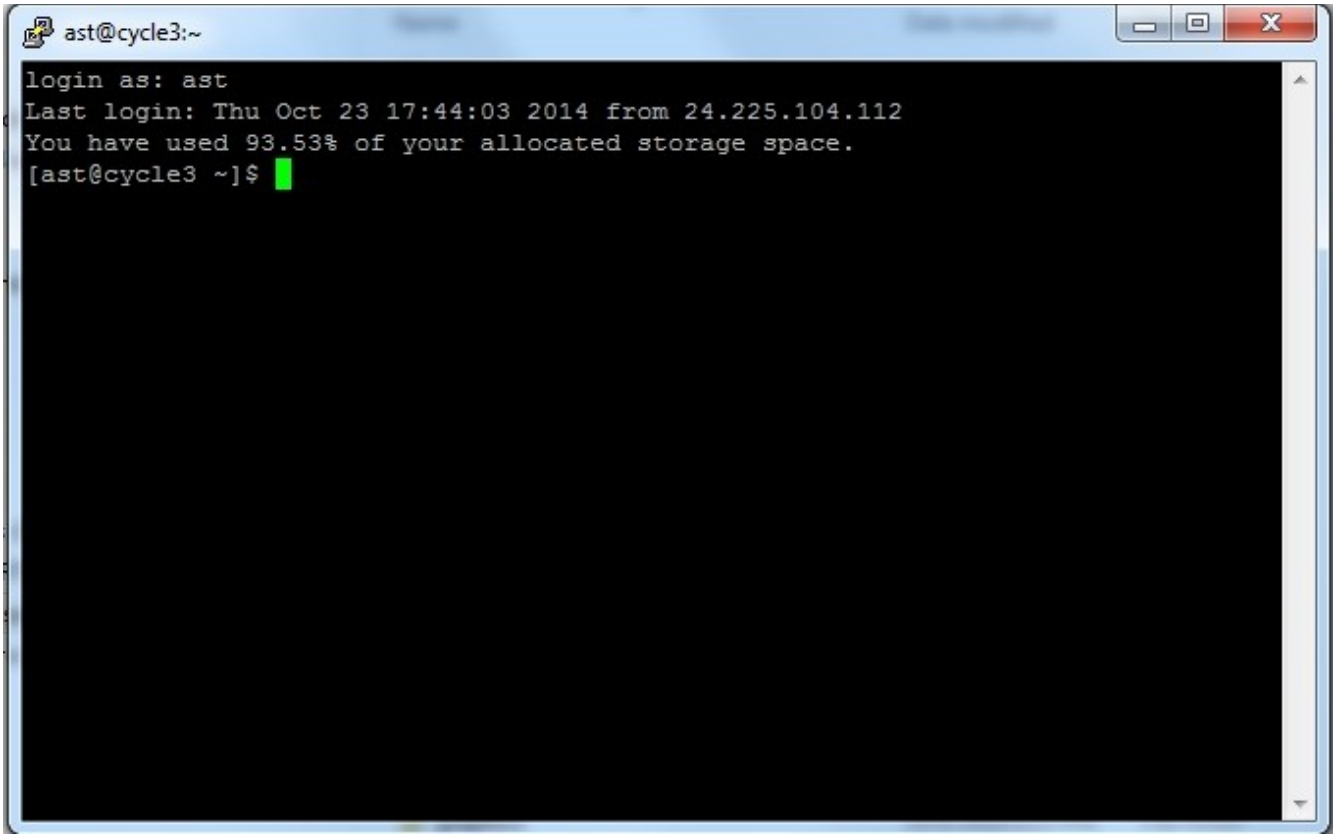

Illustration 2: Upon successful login, a terminal should be available to work in.

Upon a successful login, you should have a terminal available for use as shown in Illustration 2. The terminal allows one to work on the cycle server using the linux command line. Note that one may also work in one of the desktop computers in the one of the two linux labs.

Note: *you can access your home directory from both within the terminal, and through windows explorer. This may come in handy if you wish to inspect or modify any files while using the terminal.*

Preparing the Environment

Once you have connected to one of the cycle servers, follow the below step-by-step instructions to prepare the environment. A description of each step is provided, as well as the corresponding command. One should be able to simply copy-paste the commands given below:

1. Create a directory to work out of, called 'weka_lab'
    mkdir weka_lab

2. Make 'weka_lab' the active directory
    cd weka_lab

3. Download the java weka library
    wget http://prdownloads.sourceforge.net/weka/weka-3-7-11.zip

4. Unzip the java weka library
    unzip weka-3-7-11.zip

5. Add the java library to the java class path
    export CLASSPATH=:$(pwd)/weka-3-7-11/weka.jar:$CLASSPATH

6. Download the provided data, this should have been emailed to you as TrainData.zip. Unzip this file in your weka_lab directory, it should create a folder called “TrainData”, which contains the training data sets, as well as a test dataset with unknown labels. *The data files should now be located at weka_lab/TrainData*

Note: *Every time you create a new environment(every time you start a new terminal session), you will need to CD into the weka_lab directory and re-export the CLASSPATH variable. This is done by executing the above steps #2 and then #5.*

Using Weka from the Command Line

Once the environment is setup and the CLASSPATH variable is exported, it is possible to use the weka classifiers from the command line. To verify that everything has been set correctly, run the following command:

```
java -Xmx2012m  weka.classifiers.trees.RandomForest -h
```

This command calls java, “-Xmx2012m” specifies the maximum amount of memory to use during the execution of this java program. “weka.classifiers.trees.RandomForest” is the java class to run. “-h” is an option given to the weka.classifiers.trees.RandomForest class that requests a help page be printed. At this point you should have observed the help page for the Weka RandomForest classifier. Some of the options we will often use when calling all classifiers are:

-t <name of training file>

Sets training file.

-T <name of test file>
Sets test file. If missing, a cross-validation will be performed
on the training data.

-c <class index>
Sets index of class attribute (default: last).

-x <number of folds>
Sets number of folds for cross-validation (default: 10).

-i
Outputs detailed information-retrieval statistics for each class.

-l <name of input file>
Sets model input file. In case the filename ends with '.xml',
a PMML file is loaded or, if that fails, options are loaded
from the XML file.

-d <name of output file>
Sets model output file. In case the filename ends with '.xml',
only the options are saved to the XML file, not the model.

Additionally, near the end of the help page there are often options specific to the classifier. In the case of RandomForest, you are able to set the number of trees, numbers of features, and the depth of the forest. When training a classifier, these are the parameters that will you manipulate when tuning.

To browse the available classifiers and their options, visit:
http://weka.sourceforge.net/doc.dev/weka/classifiers/Classifier.html

Exercise 1

Step through a train and test cycle using weka on the command line.

1. Train a model using only the training dataset, store the model in the file 'forest.model'. Note that the output of this training comes in two portions, a small report of the error details when training, and also when training using stratified cross-validation. The error details of training with cross-validation are what we are interested in.

java -Xmx2012m weka.classifiers.trees.RandomForest -t TrainData/EECS_738_sr_are_train.arff -c 1 -d forest.model

2. Load the model and make predictions on the test set. This command will results in all the predictions being printed to the terminal.

java -Xmx2012m weka.classifiers.trees.RandomForest -T TrainData/ECCS_738_TestSet.arff -c 1 -l forest.model -p 0

If you wish, you can use the capabilities of the linux command line to write the predictions to a file, this is done by appending ' > myPredictions.txt' to the above command. *Note you can pipe the output to a file of any name, it doesn't have to be named "myPredictions.txt'. Note that the test dataset does not have labels, the predictions that we've made are on samples with unknown labels.*

Using Weka within Java Code.

Since Weka is written in and supplied as a java library, it is possible to call it directly from within java programs directly. I have provided a short example java program to demonstrate this. Please open *testWeka.java* for this example. The provided code is well provided, using weka within in java provides much more flexibility that the command line. For example, with a small amount of effort, one can run one or many classifiers with many different parameters. This may be useful during the project portion of this course.

Exercise 2

In this exercise, we will compile and run the java code for the J48 decision tree classifier.

1. If the java CLASSPATH variable has not been set, do so. Refer to steps #2 & #5 in the section *Preparing the Environment on page #1*.

2. CD in your "weka_lab" directory.
cd weka_lab

3. Compile the provided java code. After compilation, the file testWeka.class is produced.
javac testWeka.java

4. Run the provided java code.
java -Xmx2012m testWeka

Once the code has finished running, a file called prediction.csv file should have been produced. This file contains predictions on the test dataset in the CSV format. Note that the test dataset does not have labels, the predictions that we've made are on samples with unknown labels.

The output you should expect to see on the terminal:

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.992	0.864	0.954	0.992	0.973	0.241	0.581	0.956	I
	0.136	0.008	0.500	0.136	0.213	0.241	0.581	0.146	A
Weighted Avg.	0.947	0.819	0.930	0.947	0.933	0.241	0.581	0.913	

Mini-Project

1. Select one of the project data sets.
2. Run three classifiers using cross-validation, record the accuracy and AUC.

What to turn in:

1. Turn in a report documenting your approach to the mini-project, did you use weka on the command line or program using java?

2. The report must contain a table showing the dataset you used, the 3 classifiers, and the resulting accuracy and AUC from training using cross-validation.

3. The report must also provide a response for the following questions:

– How is accuracy calculated?

– What is stratified cross validation?

– What is AUC, how is it calculated? How is it related to the dataset of the mini-project?

– What does accuracy and AUC tell us about a classifier? In this mini-project, which is better at measuring the quality of a classifier? Why?