Lab Exercise 4

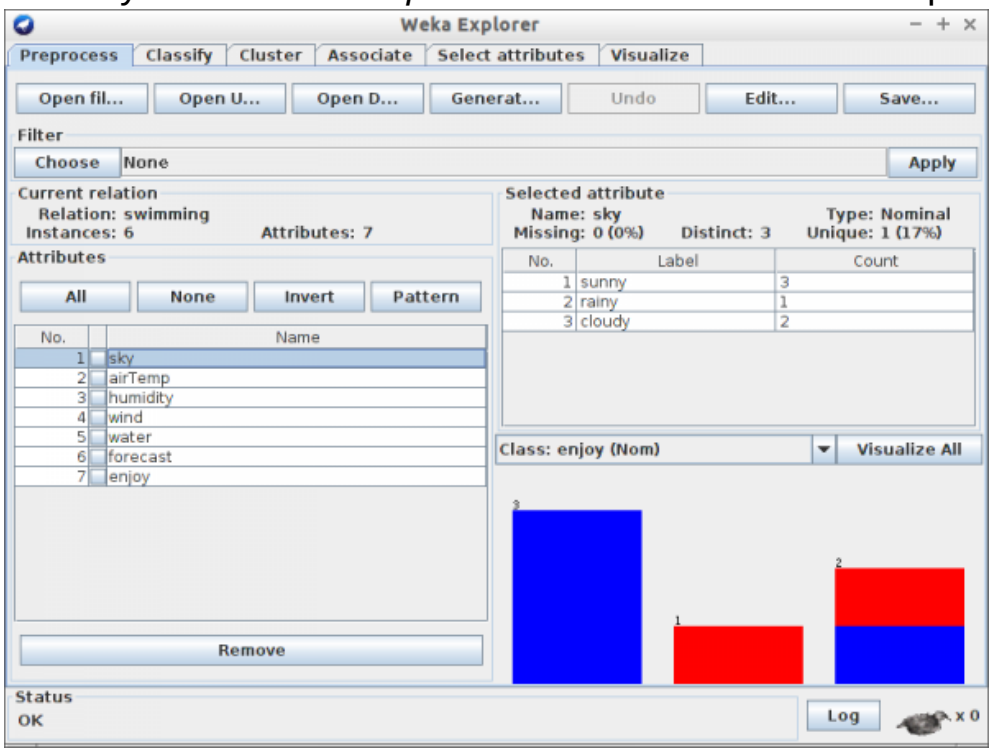
Introduction to data classification - Decision Trees & Naïve Bayes classifier

A decision tree is a graphical method of supporting the decision-making process, used in decision theory. The decision tree algorithm is also used in machine learning to generate knowledge based on given examples.

The aim of this laboratory is to use the WEKA package to generate a decision tree (decision table).

I. Data loading and analysis

1. Download data pack: data.tar.gz(: https://github.com/caiomsouza/ml-opendatasets/tree/master/weka-dataset-arff)
2. Open in a notebook (or other text editor) a file named **swimming.arff** and learn the structure of this learning file with its symbolic data vectors.
3. Start Weka, click the Explorer button and load the swimming.arff data file.
4. Analyse the first '*Preprocess*' tab and answer the questions below:



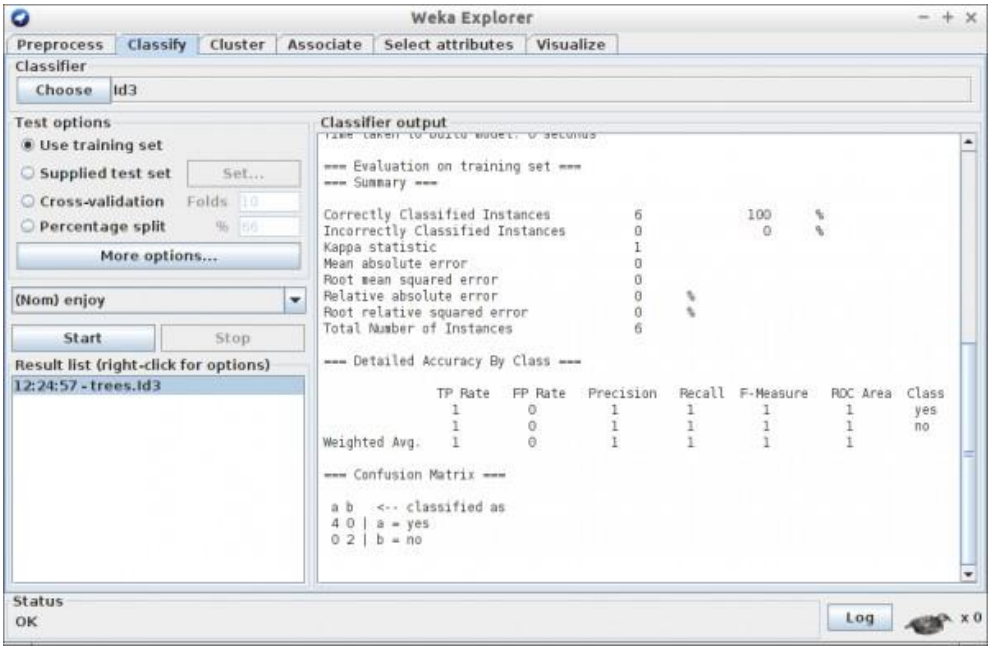
a) What is the size of the training set?
b) How many attributes exist in the training set?

c) How many instances are positive (Enjoy = yes) and how many negative?
d) Which attribute best separates the data?
e) How many elements from the data set have the humidity attribute set as high?

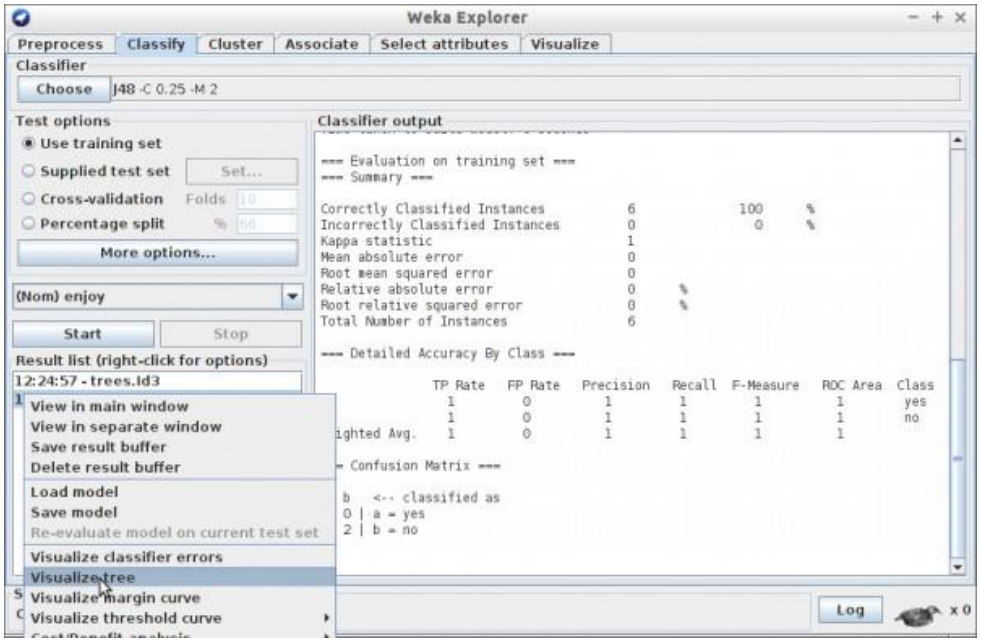
II. Load and analyse data

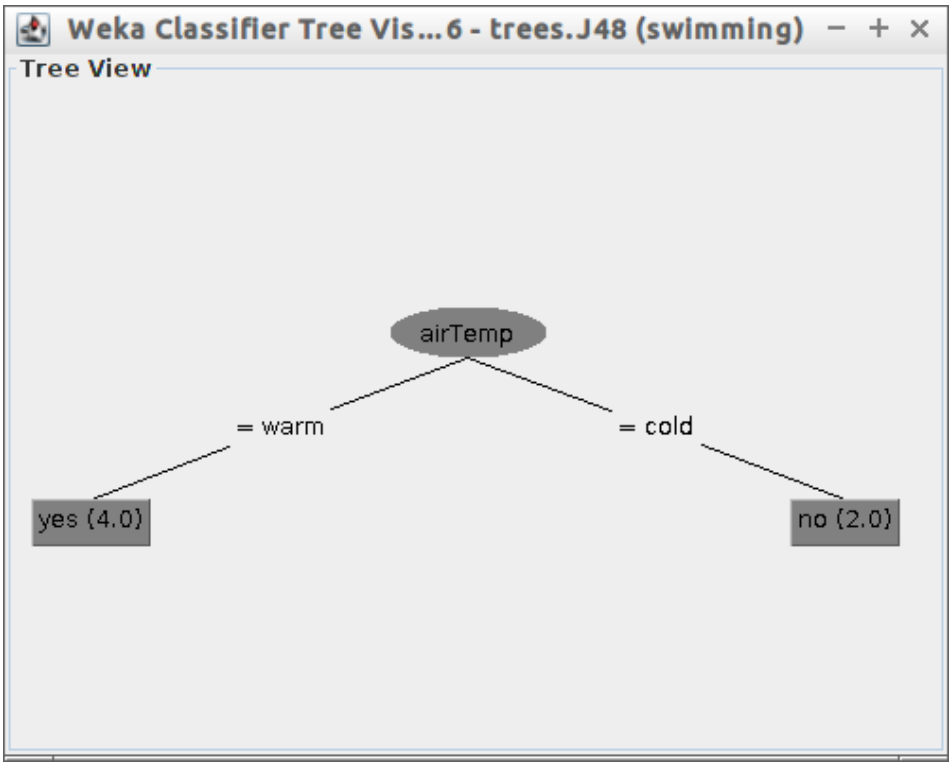
1. Open the Classify tab.
2. Select the J48 classifier using the Choose button.
3. Make sure that '*Use training'* set is checked in the '*Test options'* window. Attention! In the future, we will **not** use this form of testing - we are forced here because of the small training set.

4. Click on the Start button. Look at the result. What do the results mean?



5. Select the J48 classifier using the Choose button and click Start, thenvisualize the tree as shown below:



6. Does the tree look like this?
7. 

III. Classification accuracy

1. Load the file credit-g.arff to Weki. It contains learning data for the system, which on the basis of the attributes contained in the file, should determine whether a given set of attribute values indicates a credibility of bank customers – i.e. whether the bank should grant him a loan or if it is too risky to do so.
2. Open Classify tab and choose J48 algorithm.
3. In the '*Test options'* area, select Percentage split and type in 66%. IT means that 66% of the data will be used for learning and 34% of this data set will be used for validation.
4. Run the algorithm. How many percent of cases were correctly classified? Is this a good result?
5. Change the classifier to ZeroR from the rules branch. What is the obtained result? Better or worse than J48?
6. Try other classifiers. What are their results?
7. Go to the '*Preprocess'* tab and see how the distribution of the attribute defines whether the set is good or bad. What would be the effectiveness of an algorithm that regardless of the value of attributes would "shoot" that the user is reliable or

not?

8. Why is it worth taking a look at the data before attempting a classification task?

Naïve Bayes classifier

We will use Weka to train a Naïve Bayes classifier for the purposes of spam detection. .

The data set we will consider is the Spambase set, consisting of tagged emails from a single email account. Read through the description available for this data to get a feel for what you're dealing with. The data has been converted to ARFF format for you:

- spambase.arff

Some simple preprocessing of the data will be required before it is ready for use. We can do this in Weka:

1. From the *Preprocess* (default) tab in Weka, hit *Open file...* and select the spambase.arff file that you downloaded above.
2. A full list of the attributes in this data set will appear in the "Attributes" frame.
3. Delete the capital_run_length_average, capital_run_length_longest and capital_run_length_total attributes by checking the box to their left and hitting the *Remove* button.
4. The remaining attributes represent relative frequencies of various important words and characters in emails. We wish to convert these to Boolean values instead: 1 if the word or character is present in the email, 0 if not. To do this, select the *Choose* button in the *Filter* frame at the top of the window, and pick *filters* > *unsupervised* > *attribute* > *NumericToBinary*. Now hit the *Apply* button. All the numeric frequency attributes are now converted to Booleans. Each e-mail is now represented by a 55 dimensional vector representing whether or not a particular word exists in an e-mail. This is the so called *bag of words* representation (this is clearly a very crude assumption since it does not take into account the order of the words).
5. Save this preprocessed data set for future use using the *Save...* button.

Given the data set we've just loaded, we wish to train a Naïve Bayes classifier to distinguish spam from regular email by fitting a distribution of the number of occurrences of each word for all the spam and non-spam e-mails. Under the *Classify* tab:

1. Select *Choose* in the *Classifier* frame at the top and select *classifiers* > *bayes* > *NaiveBayes*.
2. Leave the default settings and hit *Start* to build the classifier. Study the output produced, most importantly the percentages of correctly and incorrectly classified instances. You probably will notice that your classifer does rather well despite making a very strong assumption on the form of the data.
    - Can you come up with a reason for the good performance? What would be the main practical problems we would face if we were not to make this assumption for this particular dataset?
    - How long did your classifer take to train and classify? Given this, how scalable do you think the Naïve Bayes classifier is to large datasets? Can you come up with a good reason for this?

3. Examine the classifier models produced by Weka (printed above the performance summary). Find the prior probabilities for each class.

- How does Naïve Bayes compute the probability of an e-mail belonging to a class (spam/not spam)?
- Compute the conditional probability of observing the word "3d" given that an e-mail is spam P(3d|spam) and that it is non-spam P(3d|non-spam). To do this, we need to use the counts of the built model that are produced within the *Classifier output* screen under the *Classify* tab. The general format of the Weka count output is (*Note:* this is a toy example. You will need to examine your Weka output to find the true counts for the word "3d".):

	Class	
	0	***1***
0	1	2
1	3	4
total	4	6

- This means that 4 instances (e.g. e-mails) contain that particular attribute value (e.g. the word "3d") in Class 1 (e.g. Is Spam). 2 instances didn't contain that value of the attribute in Class 1. 3 instances of Class 0 contained that attribute value, whilst 1 instance of Class 0 (e.g. Not Spam) didn't contain that attribute value. The totals reflect the number of instances belonging to both classes e.g. the number of e-mails that are Spam and not Spam.

4. For the final part of this section we will now pretend we are spammers wishing to fool a spam checking system based on Naïve Bayes into classifying a spam e-mail as ham (i.e. a valid e-mail). We will now use all of the training data to train our classifier and apply the learnt classifer to a dedicated test set. Load the test set in Weka. Under the *Classify tab*, select *supplied test set > set > open file* and set the test file to the supplied spambase_test.arff. This ARFF file contains the binary vector representing one spam e-mail. Run the Naïve Bayes classifer on this test set. Does the classifer classify the spam e-mail correctly?
5. Open the test file spambase_test.arff in emacs or another text editor. Identify good non-spam words and add these to the e-mail. *Important:* Leave the class label (last attribute value) in the test data file untouched. During testing, Weka will ignore this attribute and will instead use our previously trained classifier to predict the class label of this e-mail. Re-run the classifer on the modified test set. Has the class label (spam/non-spam) for this e-mail changed?

   You've now managed to switch the predicted class label for that e-mail. Adding more “hammy” words to this e-mail has sufficiently increased the probability that this e-mail is ham so the classifier now outputs "ham" as the e-mail's class label (by changing the word content of the e-mail you have added extra evidence or "votes" towards this e-mail being classified as ham). This is the "stuffing" example given in the lectures and is directly caused by the independence assumption that is made by Naive Bayes. Each word contributes independently of each other to the final score. This is a reason that a lot of spam e-mails include random excerpts from the passages of books so as to effectively add “ hammy” words in the hope

that the spam e-mail will bypass the spam filters. For this reason, in practice, many commercial e-mail systems (consider Gmail) likely use a lot more sophisticated spam detection models.