**IAML LAB 1: NAÏVE BAYES AND DECISION TREE CLASSIFICATION**

In this lab we introduce [Weka](http://www.cs.waikato.ac.nz/ml/weka/), an open source package for data mining and machine learning. We work with two data sets, a spam filtering dataset, and a credit scoring dataset and we perform classification tasks using Naive Bayes and Decision trees.

**WEKA**

1. On DICE, use weka to get the application running, and select the *Explorer* button in the opening splash window.

***However, Weka may not yet be installed on DICE SL7!****If this doesn't work, go to Applications>Education and see if Weka appears there - if it does launch it from there   
  
If that fails, try executing:   
  
/afs/inf.ed.ac.uk/user/n/ngoddard/runweka   
  
And if that fails, try executing:   
  
java -Xmx1000M -jar /afs/inf.ed.ac.uk/user/n/ngoddard/weka/weka-3-6-13/weka.jar   
  
And if that fails, email the instructor and the TAs!*

1. Documentation for Weka is available from its [website](http://www.cs.waikato.ac.nz/ml/weka/). For this lab, the instructions will be explicit to get you acquainted with the system.
2. Weka uses the [ARFF data format](http://www.cs.waikato.ac.nz/ml/weka/arff.html), a very simple text format. For real applications it is usually necessary to convert the original data into ARFF format so that it may be opened in Weka. Perl is generally good for this. For these labs this is done for you, although it may be an important consideration in your own work.
3. Occasionally you may find that Weka crashes with an out of memory exception, particularly for larger data sets or extensive preprocessing of data. This is related to the default heap size allocation for the Java Virtual Machine (JVM). To run Weka with a larger memory allocation, download the following script: [[runweka.sh](http://www.inf.ed.ac.uk/teaching/courses/iaml/lab/runweka.sh)] and execute the command sh runweka.sh. You can control memory allocation with the -Xmx option in the script, for example -Xmx2048m would allocate up to 2GB of memory to Weka. Please note that this may slow down your machine.

**SPAM FILTERING**

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

The data set we will consider is the [Spambase](http://archive.ics.uci.edu/ml/datasets/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](http://www.inf.ed.ac.uk/teaching/courses/iaml/lab/data/spambase.arff)

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

1. Familiarize yourself with the [ARFF format](http://weka.wikispaces.com/ARFF+%28stable+version%29)
2. From the *Preprocess* (default) tab in Weka, hit *Open file...* and select the spambase.arff file that you downloaded above.
3. A full list of the attributes in this data set will appear in the "Attributes" frame.
4. 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.
5. 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*](http://en.wikipedia.org/wiki/Bag_of_words_model) representation (this is clearly a very crude assumption since it does not take into account the order of the words).
6. Save this preprocessed data set for future use using the *Save...* button. You will need this for lab 2.

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.

1. 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](http://www.inf.ed.ac.uk/teaching/courses/iaml/lab/data/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?
2. Open the test file [spambase\_test.arff](http://www.inf.ed.ac.uk/teaching/courses/iaml/lab/data/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.

**DECISION TREES**

One of the great advantages of decision trees is their *interpretability*. The rules learnt for classification are easy for a person to follow, unlike the opaque "black box" of many other methods, such as neural networks. We demonstrate the utility of this using a [German credit](http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29) data set. You can read [a description](http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29) of this dataset at the UCI site. The task is to predict whether a loan approval is good or bad credit risk based on 20 attributes. We've simplified the data set somewhat, particularly making attribute names and values more meaningful.

1. Download the [credit.arff](http://www.inf.ed.ac.uk/teaching/courses/iaml/lab/data/credit.arff) dataset and load it to Weka.
2. When presented with a dataset, it is usually a good idea to visualise it first. Go to the *Visualise* tab. Click on any of the scatter plots to open a new window which shows the scatter plot for two selected attributes. Try visualising a scatter plot of *age* and *duration*. Do you notice anything unusual? You can click on any data point to display all it's values.
3. In the previous point you should have found a data point, which seems to be corrupted, as some of its values are nonsensical. Even a single point like this can significantly affect the performance of a classifier. How do you think it would affect Decision trees? How about Naive Bayes? A good way to check this is to test the performance of each classifier before and after removing this datapoint.
4. To remove this instance from the dataset we will use a filter. We want to remove all instances, where the age of an applicant is lower than 0 years, as this suggests that the instance is corrupted. In the *Preprocess* tab click on *Choose*in the Filter pane. Select *filters > unsupervised > instance > RemoveWithValues*. Click on the text of this filter to change the parameters. Set the attribute index to 13 (Age) and set the split point at 0. Click *Ok* to set the parameters and *Apply* to apply the filter to the data. Visualise the data again to verify that the invalid data point was removed.
5. On the *Classify* tab, select the *Percentage split* test option and change its value to 90%. This way, we will train the classifiers using 90% of the training data and evaluate their performance on the remaining 10%.
6. First, train a decision tree classifier with default options. Select *classifiers > trees > J48* and click *Start*. *J48* is the Weka implementation of the *C4.5* algorithm, which uses the normalized information gain criterion to build a decision tree for classification.
7. After training the classifier, the full decision tree is output for your perusal; you may need to scroll up for this. The tree may also be viewed in graphical form by right-clicking in the *Result list* and selecting *Visualize tree*; unfortunately this format is very cluttered for large trees. Such a tree accentuates one of the strengths of decision tree algorithms: they produce classifiers which are understandable to humans. This can be an important asset in real life applications (people are seldom prepared to do what a computer program tells them if there is no clear explanation). Observe the output of the classifier and try to answer the following questions:
   * How would you assess the performance of the classifier? Is the Percentage of *Correctly Classified Instances* a sufficient measure in this case? Why? *Hint:* check the number of good and bad cases in the test sample, using the confusion matrix. Each column of the matrix represents the instances in a predicted class, while each row represents the instances in an actual class. For example let us define an experiment from P positive instances and N negative instances. The four outcomes can be formulated in a 2 by 2 contingency table or confusion matrix, the breakdown of which is given below:

|  |  |  |  |
| --- | --- | --- | --- |
|  | | **actual value** | |
|  | | ***p*** | ***n*** | **total** |
| **prediction outcome** | ***p'*** | True Positive | False Positive | P' |
| ***n'*** | False Negative | True Negative | N' |
| **total** | | P | N |  |

* + One benefit of a confusion matrix is that it is easy to see if the system is confusing two classes (i.e. commonly mislabeling one as another).
  + Looking at the decision tree itself, are the rules it applies sensible? Are there any branches which appear absurd? At what depth of the tree? What does this suggest?   
    *Hint:* Check the rules applied after following the paths: (a) *CheckingAccount = <0, Foreign = yes, Duration >11, Job = skilled, OtherDebtors = none, Duration <= 30* and (b)  *CheckingAccount = <0, Foreign = yes, Duration >11, Job = unskilled.*
  + How does the decision tree deal with classification in the case where there are zero instances in the training set corresponding to that particular path in the tree (e.g. those leaf nodes that have (0:0))?

1. Now, explore the effect of the *confidenceFactor* option. You can find this by clicking on the Classifer name (to the right of the *Choose* button on the Classify tab). On the *Classifier options* window, click on the *More* button to find out what the confidence factor controls. Try the values 0.1, 0.2, 0.3 and 0.5. What is the performance of the classifier at each case? Did you expect this given your observations in the previous questions? Why do you think this happens?
2. Suppose that it is worse to classify a customer as good when they are bad, than it is to classify a customer as bad when they are good. Which value would you pick for the confidence factor? Which performance measure would you base your decision on?
3. *If you have time:* Finally we will create a *random decision forest* and compare the performance of this classifier to that of the decision tree and the decision stump. The random decision forest is an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the class's output by individual trees. Again set the test option *Percentage split* to 90%. Select *classifiers > trees > RandomForest* and hit *Start*. Again, observe the output. How high can you get the performance of the classifier by changing the number of trees (numTrees) parameter? How does the random decision forest compare performance wise to the decision tree and decision stump?