F21DL Data Mining and Machine Learning: Coursework Assignment 3
Handed Out: Thursday 26th October 2012
What must be submitted: A report of maximum 3sides of A4, in PDF format
To be 'Handed in': 23:59pm Friday November 23rd 2012
-- by email to dwcorne@gmail.com with Subject Line: DMML Coursework 3
Worth: 30% of the marks for the module**.**

The point: this coursework is designed to give you experience with, and hence more understanding of:

- Overfitting: finding a classifier that does very well on your training data doesn't mean it will do well on unseen (test) data.
- The relationship between overfitting and complexity of the classifier – the more degrees of freedom in your classifier, the more chances it has to overfit the training data.
- The relationship between overfitting and the size of the training set.
- Bespoke machine learning: you don't have to just use one of the standard types of classifier – the 'client' may specifically want a certain type of classifier (here, a ruleset that works in a certain way), and you can develop algorithms that try to find the best possible such classifier.

In this coursework you will work with only the "Optical recognition of handwritten digits" dataset from the UCI repository. You will use the following specific training and testing sets, which I have prepared:
http://www.macs.hw.ac.uk/~dwcorne/Teaching/DMML/optraining.txt
http://www.macs.hw.ac.uk/~dwcorne/Teaching/DMML/optesting.txt

and you will use this C program, which learns a *ruleset*:
http://www.macs.hw.ac.uk/~dwcorne/Teaching/DMML/ers.c

The raw data of the 'optical recognition' dataset came from 32x32 pixellated images of handwritten digits, (e.g. either '0', '1', '2', etc…, '9') . where each pixel was either black or white. These were preprocessed to become 8x8 arrays, where each pixel represented a 4x4 square from the raw data, and hence is a number between 0 and 16 inclusive. This is the data you see in the above files. Each instance gives the 64 pixels in row by row order, with the last field (the 65th field) indicating the digit that is represented in this image.

The C program `ers.c` produces a ruleset of the following form:

```
IF  (Field27 is between 2 and 8)    AND (Field44 is between 0 and 14)  THEN class = 0
IF  (Field01 is between 12 and 15)  AND (Field07 is between 5 and 12)  THEN class = 0
IF  (Field19 is between 3 and 8)    AND (Field29 is between 4 and 14)  THEN class = 1
IF  (Field11 is between 1 and 12)   AND (Field21 is between 8 and 9)   THEN class = 1
… etc …
IF  (Field04 is between 0 and 3)    AND (Field24 is between 4 and 4)   THEN class = 9
IF  (Field01 is between 10 and 16)  AND (Field27 is between 0 and 12)  THEN class = 9
```

When you run the program, you fix `fieldsperrule` at the command line (the number of fields AND'ed together in each rule – this can vary between 1 and 64), and you fix `rulesperclass` at the command line, which can vary between 1 and 50.

When this ruleset is used to classify an instance, what happens is this: each rule in the ruleset is tried in turn. if everything on the LHS of a rule (before the 'THEN') matches the instance, then the ruleset records 1 'vote' for the class on the RHS of that rule. When each rule has been processed, there are a number of votes for each class. The class with most votes is predicted as the class of the new instance. However if there is a tie for the most votes, then the ruleset is saying 'don't know', and this counts as misclassification.

When you run the program, as you will see, it prints out the best so far accuracy on the training set after every 100 iterations. At the end, it prints on screen the best ruleset on the training set, and shows you its accuracy on both the training and the test data. The algorithm that tries to find a good ruleset is stochastic, and this program will give a different result every time you rerun the same experiment.

What to do

Get hold of the training and test data, and the C program for evolving rulesets, as pointed to previously:

1. compile the C program on a linux or unix machine, like this:

```
gcc ers.c –o ers -lm
```

2. do a quick test run to see it works OK – like this:

```
./ers 65 100 5120 10 optraining.txt optesting.txt 10 20 10000
```

The arguments, which have to be in order, are:

- Number of fields (65)
- Number of instances in the training set (100)
- Number of instances in the testing set (5120)
- Number of classes (10)
- Training data file (optraining.txt)
- Testing data file (optesting.txt)
- **Number of fields per rule (anything from 1 to 64)**
- **Number of rules per class (anything from 1 to 50)**
- Number of iterations(keep at 10,000)

In this coursework, I only expect you to vary the green ones.

3. Evaluate the performance of ers.c for 16 combinations of fields-per-rule and rules-per-class (i.e. using 4 different values for each of those two parameters).

4. make new training and testing sets, by moving 900 of the instances in 'optesting/txt' into the training set. Then, repeat step 3.

5. make new training and testing sets again, this time with 2000 instances in the training set, and the remainder in the test set, and again repeat step 3.

What to Submit

What you submit for this assignment is a report of maximum THREE sides of A4, containing the following.

1. up to half a page describing how you did steps 3, 4, 5 – i.e. did you make up scripts to run your experiments? And how did you make the new training and test sets?
2. up to 1 page with a graphical and/or tabular description of the basic results from steps 3, 4 and 5.
3. Write three sections, respectively entitled:
   - "Variation in performance with number of fields per rule"
   - "Variation in performance with number of rules per class"
   - "Variation in performance with size of training set"

   taking up, all taken together, no more than about 1 and a half pages.
   In each of these sections you will speculate on the reasons that might underpin the performance variations that you see, considering general issues and also issues pertaining to this specific task.

The material for parts 1—3 must be all contained within 3 sides of A4. 20 marks are lost for every extra page, even if there is just one word on the page.

Marking: worth 30% of the module; of that 40%, the above parts break down as follows: 1(2), 2(12), 3(16).