COMP 307 — *Introduction to AI*

Assignment 1: Basic Machine Learning Algorithms

15% of Final Mark — Due: 11:59pm Friday 7 April 2017

1 Objectives

The goal of this assignment is to help you understand the basic concepts and algorithms of machine learning, write computer programs to implement these algorithms, use these algorithms to perform classification tasks, and analyse the results to draw some conclusions. In particular, the following topics should be reviewed:

- Machine learning concepts,
- Machine learning common tasks, paradigms and methods/algorithms,
- Nearest neighbour method for classification,
- Decision tree learning method for classification,
- Perceptron/linear threshold unit for classification, and
- k-means method for clustering and k-fold cross validation for experiments.

These topics are (to be) covered in lectures 04–07. The text book and online materials can also be checked.

2 Question Description

Part 1: Nearest Neighbour Method [30 marks]

This part is to implement the (K-)Nearest Neighbour algorithm, and evaluate the method on the *iris* data set to be described below. Addition questions on k-means and k-fold cross validation need to be answered and discussed.

Problem Description

The *iris* data set is taken from the UCI Machine Learning Repository (`http://mlearn.ics.uci.edu/MLRepository.html`), but some changes are made for this assignment. The original data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. There are four numeric attributes: *sepal length in cm, sepal width in cm, petal length in cm*, and *petal width in cm*. More detailed information can be seen from the file `iris.names`.

Requirements

Your program should classify each instance in the test set `iris-test.txt` according to the training set `iris-training.txt`.

Your program should take two file names as command line arguments, and classify each instance in the test set (the second file name) according to the training set (the first file name).

You may write the program code in `Java, C/C++`, or any other programming language.

You should submit the following files electronically and also a report in hard copy.

- (15 marks) `Program code` for your (k-)Nearest Neighbour Classifier (both the source code and the executable program running on ECS School machines), and
- (15 marks) `A report` in any of the PDF, text or DOC formats. The report should include:
    1. Report the class labels of each instance in the test set predicted by the basic nearest neighbour method (k=1), and the classification accuracy on the test set of the basic nearest neighbour method;

2. Report the classification accuracy on the test set of the k-nearest neighbour method where k=3, and compare and comment the performance of the two classifiers (k=1 and k=3);
3. Discuss the main advantages and disadvantages of this classification method.
4. Assuming that you are asked to apply the k-fold cross validation method for the above problem and *k=5*, what would you do? State the major steps.
5. In the above problem, assuming that the class labels are not available in the training set and the test set, and that there are three cluters, which method would you use to group the examples in the data set? State the major steps.
6. (Optional, bonus 5 marks) Implement the clustering method above.

Part 2: Decision Tree Learning Method [35 marks]

This part involves writing a program that implements a simple version of the Decision Tree (DT) learning algorithm, report the results, and make a number of discussions.

Problem Description

The main data set for the DT program is in the files `hepatitis.dat, hepatitis-training.dat`, and `hepatitis-testing.dat`. It describes 137 cases of patients with hepatitis, along with the outcome. Each case is specified by 16 Boolean attributes describing the patient and the results of various tests. The goal is to be able to predict the outcome based on the attributes. The first file contains all the 137 cases; the training file contains 110 of the cases (chosen at random) and the testing file contains the remaining 27 cases.

The data files are formatted as tab-separated text files, containing two header lines, followed by a line for each instance.

- The first line contains the names of the two classes.
- The second line contains the names of the attributes.
- Each instance line contains the class name followed by the values of the attributes ("true" or "false").

This data set is taken from the UCI Machine Learning Repository `http://mlearn.ics.uci.edu/MLRepository.html`. It consists of data about the prognosis of patients with hepatitis. This version has been simplified by removing some of the numerical attributes, and converting others to booleans by an arbitrary division of the range.

The file `golf.data` is a smaller data set in the same format that may be useful for testing your programs while you are getting them going. Each instance describes the weather conditions that made a golf player decide to play golf or to stay at home. The data set is not large enough to do any useful evaluation.

Decision Tree Learning Algorithm

The basic algorithm for building decision trees from examples is straightforward. Complications arise in handling multiple kinds of attributes, doing the statistical significance testing, pruning the tree, etc. but your program don't have to deal with any of the complications.

For the simplest case of building a decision tree for a set of instances with boolean attributes (yes/no decisions), with no pruning, the algorithm is shown below.

```
instances: the set of training instances that have been classified to the node being constructed;
attributes:: the list of attributes that were not used on the path from the root to this node.

BuildTree (Set instances, List attributes)
   if  instances is empty
      return a leaf node containing the name and probability of the overall
               most probable class (ie, the ''baseline'' predictor)
   if  instances are pure
      return a leaf node containing the name of the class of the instances
               in the node  and probability 1
   if  attributes is empty
      return a leaf node containing the name and probability of the
               majority class of the instances in the node (choose randomly
               if classes are equal)
   else find best attribute:
      for each attribute
         separate instances into two sets:
              instances for which the attribute is true,  and
              instances for which the attribute is false
         compute purity of each set.
          if weighted average purity of these sets is best so far
              bestAtt = this attribute
              bestInstsTrue = set of true instances
              bestInstsFalse = set of false instances
      build subtrees using the remaining attributes:
         left = BuildTree(bestInstsTrue, attributes - bestAtt)
         right = BuildTree(bestInstsFalse, attributes - bestAttr)
      return Node containing (bestAtt, left, right)
```

To apply a constructed decision tree to a test instance, the program will work down the decision tree, choosing the branch depending on the value of the relevant attribute in the instance, until it gets to a leaf. It then returns the class name in that leaf.

Requirements

Your program should take two file names as command line arguments, construct a classifier from the training data in the first file, and then evaluate the classifier on the test data in the second file.

You can write your program in Java, C/C++, or any other programming language.

You should submit the following files electronically and also a report in hard copy.

- (20 marks) `Program code` for your decision tree classifier (both source code and executable program running on the ECS School machines). The program should print out the tree in a human readable form (text form is fine).

  You may use either of the purity measures presented in the lectures. The file `helper-code.java` contains java code that may be useful for reading instance data from the data files. You may use it if you wish, but you may also write your own code.

- (15 marks) A report in PDF or text format. The report should include:

  1. You should first apply your program to the `hepatitis-training.dat` and `hepatitis-test.dat` files and report the classification accuracy in terms of the fraction of the test instances that it classified correctly. Report the learned decision tree classifier. Compare the accuracy of your Decision Tree program to the baseline classifier (which always predicts the most frequent class in the dataset), and comment on it.
  2. You should then construct 10 other pairs of training/test files, train and test your classifiers on each pair, and calculate the average accuracy of the classifiers over the 10 trials.

     There is a script `split-datafile` that takes the name of the full data set (eg, `hepatitis`), the number of training instances, and a suffix for the filenames, and will construct pairs of training and test files. For example

     `./split-datafile hepatitis.dat 100 run1`

     will construct the files `hepatitis-training-run1.dat` and `hepatitis-test-run1.dat` with 100 and 37 instances respectively.

3. "Pruning" (removing) some of leaves of the decision tree would always make the decision tree less accurate on the training set. Explain (a) How you could prune leaves from the decision tree; (b) Why it would reduce accuracy on the training set, and (c)Why it might improve accuracy on the test set.
4. Explain why the impurity measure is not a good measure if there are three or more possible classes that the decision tree must distinguish.

Note: A Simple way of Outputting a Learned Decision Tree

The easiest way of outputting the tree is to do a traversal of the tree. For each non-leaf node, print out the name of the attribute, then print the left tree, then print the right tree. For each leaf node, print out the class name in the leaf node and the fraction. By increasing the indentation on each recursive call, it becomes somewhat readable.

Here is a sample tree (not a correct tree for the golf dataset). Note that the final leaf node which is impure can only occur on a path which has used all the attributes so there is no attribute left to split the instances any further. This should not happen for the data set above, but might happen for some datasets.

```
cloudy = True:
    raining = True:
        Class StayHome, prob = 1.0
    raining = False:
        Class PlayGolf, prob = 1.0
cloudy = False:
    hot = true:
        Class PlayGolf, prob = 1.0
    hot = False:
           windy = True:
                Class StayHome, prob = 1.0
           windy = False:
                Class PlayGolf, prob = 0.75
```

Here is some sample (Java) code for outputting a tree.

In class `Node` (a non-leaf node of a tree):

```
public void report(String indent){
  System.out.format("%s%s = True:\n",
                    indent, attName);
  left.report(indent+"    ");
  System.out.format("%s%s = False:\n",
                    indent, attName);
  right.report(indent+"    ");
 }
```

In class `Leaf` (a leaf node of a tree):

```
public void report(String indent){
 if (count==0)
   System.out.format("%sUnknown\n", indent);
 else
   System.out.format("%sClass %s, prob=$4.2f\n",
              indent, className, probability);
 }
```

Part 3: Perceptron [35 marks]

This part of the assignment involves writing a program that implement a perceptron with random features that learns to distinguish between two classes of black and white images (X's and O's).

Data Set

The file `image.data` consists of 100 image files, concatenated together. The image files are in `PBM` format, with exactly one comment line.

- The first line contains `P1`;
- The second line contains a comment (starting with `#`) that contains the class of the image (`Yes`, or `Other`),
- The third line contains the width and height (number of columns and rows)
- The remaining lines contain 1's and 0's representing the pixels in the image, with '1' representing black, and '0' representing white. The line breaks are ignored.

Note that the PBM image format rules are a little more flexible than this. The image files in the data set do not have any comments other than the one on the second line but the PBM rules allow comments starting with `#` at any point in the file after the first token. In the image files in the data set, the pixels have no whitespace between them (other than the line breaks), but PBM rules also allow the 1's and 0's to be separated by white space. However, the PBM rules do not allow image files to be concatenated into a single file!

Features

The program should construct a perceptron that uses at least 50 random features. Each feature should be connected to 4 randomly chosen pixels. Each connection should be randomly marked as *true* or *false*, and the feature should return 1 if at least three of the connections match the image pixel, and return 0 otherwise.

For example, if the Feature class has fields:

```
class feature {
  int[] row;
  int[] col;
   boolean[] sgn;
```

then a feature with positive connections to pixels (5,2), and (2,9), and negative connections to (6,5) and (9,1) could be represented using three arrays of length 4:

```
f.row = { 5, 2, 6, 9};
f.col = { 2, 9, 5, 1};
f.sgn = { true, true, false, false};
```

and the value of the feature on an image (represented by a 2D boolean array) could be computed by

```
int sum=0;
for(int i=0; i < 4; i++)
    if (image[f.row[i], f.col[i]]==f.sgn[i]) sum++;
return (sum>=3)?1:0;
```

You may find it convenient to calculate the values of the features on each image as a preprocessing step, before you start learning the perceptron weights. This means that each image can be represented by an array of feature values. Don't forget to include the "dummy" feature whose value is always 1.

If you are using Java, `new Java.util.Random()` will create a random number generator with a `nextInt(int n)` method that returns an int between 0 and n−1, and a `nextBoolean(int n)` method that returns a random boolean. To get the same sequence of random numbers every time, you can call the constructor with an integer (or long) argument that sets the seed of the random number generator to start at the same state each time.

Simple Perceptron Algorithm

A perceptron with n features is represented by a set of real valued weights, $\{w_0, w_1, ..., w_n\}$, one weight for the threshold, and one for each feature. Given an instance with features $f_1, ..., f_n$, the perceptron will classify the instance as a positive instance (a member of the class) only if

$$\sum_{i=0}^{n} w_i f_i > 0$$

where f_0 is a "dummy" feature that is always 1.

The algorithm for learning the weights of the perceptron was given in lectures:

```
Until the perceptron is always right (or some limit):
    Present an example (+ve or -ve)
    If perceptron is correct, do nothing
    If -ve example and wrong
        (ie, weights on active features are too high)
        Subtract feature vector from weight vector
    If +ve example and wrong
        (ie, weights on active features are too low)
        Add feature vector to weight vector
```

Your program should implement this algorithm, using the feature vectors calculated from the images using the feature detectors. It should present the whole sequence of training examples, several times over, until either the perceptron is correct on all the training examples or else it is not converging (eg, it has presented all the examples 1000 times). It should then use the perceptron to classify all the examples.

Useful file

The file `MakeImage.java` would let you construct your own image files if you wish (the data file `image.data` has been created using this program). More useful is the `load` method in that file that will read a `pbm` file (as long as it has exactly the one comment and no spaces between the pixels). You may modify this to load the data from `image.data`.

Requirements

The program should take one file name (the image data file) as a command line argument, then

- load the set of images from the data file
- construct a set of random features
- construct a perceptron that uses the features as inputs
- train the perceptron on the images until either it has presented the whole set of images at least 100 times or the perceptron weights have converged (ie, the perceptron is correct on all the images).
- report on the number of training cycles to convergence, or the number of images that are still classified wrongly.
- print out the random features that it created, and the final set of weights the perceptron acquired.

You can write your program in Java, C/C++, or any other programming language.

In this part of the assignment, you should submit:

- (25 marks) `Program code` for your perceptron classifier (both source code and executable program running on the ECS School machines).
- (10 Marks) A report in any of the PDF, text or DOC formats. The report should include:
    1. Report on the performance of your perceptron. For example, did it find a correct set of weights?
    2. Explain why evaluating the perceptron's performance on the training data is not a good measure of its effectiveness. You may wish to create additional data to get a better measure. If you do, report on the perceptron's performance on this additional data.

3 Relevant Data Files and Program Files

The relevant data files, information files about the data sets, and some utility programs files can be found from the following diretory:

`/vol/comp307/assignment1/`

Under this directory, there are four subdirectories `part1, part2`, and `part3` corresponding to the four parts of the assignment, respectively.

4 Assessment

We will endeavour to mark your work and return it to you as soon as possible, hopefully in 2 weeks. The tutor(s) will run a number of helpdesks to provide assistance.

5 Submission Guidelines

5.1 Submission Requirements

1. Programs for all individual parts. To avoid confusion, all the individual parts should use directories `part1/, part2/, ...` and all pieces of programs should be stored in their corresponding directories. Within each directory, please provide a `readme file` that specifies how to compile and run your program. A script file called `sampleoutput.txt` should also be provided to show how your program run properly. If you programs cannot run properly, you should provide a `buglist` file.
2. A document that consisting of the report of all the individual parts. The document should mark each part clearly. The document can be written in PDF, text or the DOC format.

5.2 Submission Method

The programs and the PDF version of the document should be submitted through web submission system from the COMP307 course web site **by the due time**.

The hard copy of the document (excluding the program code) is *also* required to submit to the COMP 307 handin box in the 2nd floor corridor of the Cotton building by the due time.

5.3 Late Penalties

The assignment must be handed in on time unless you have made a prior arrangement with the lecturer or have a valid medical excuse (for minor illnesses it is sufficient to discuss this with the lecturer). The penalty for assignments that are handed in late without prior arrangement is one grade reduction per day. Assignments that are more than one week late will not be marked.