COMP 307/AIML 420 — *Introduction to* AI

Assignment 1: Basic Machine Learning Algorithms

15% of Final Mark — Due: 11:59pm Wednesday 30 March 2022

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, you should be familiar with the following topics:

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

These topics are (to be) covered in lectures 4–7. The textbook and online materials can also be checked.

2 Question Description

Part 1: k-Nearest Neighbour Method (30 Marks for COMP307, and 37 Marks for AIML420)

In this part you will implement the k-Nearest Neighbour method, and evaluate it on the *wine* data set described below. Additional questions on k-means and k-fold cross validation need to be answered/discussed.

Problem Description

The *wine* data set is taken from the UCI Machine Learning Repository (`https://archive.ics.uci.edu/ml/datasets/wine`). The data set contains 178 instances in 3 classes, having 59, 71 and 48 instances, respectively. Each instance has 13 attributes: *Alcohol, Malic_acid, Ash, Alcalinity_of_ash, Magnesium, Total_phenols, Flavanoids, Nonflavanoid_phenols, Proanthocyanins, Color_intensity, Hue, OD280/OD315_of_diluted_wines*, and *Proline*. We have split the dataset into two subsets: one for training and the other for testing.

Requirements

Your program should classify each instance in the test set `wine-test` according to the training set `wine-training`. Note that the *final* column in these files list the class label for each instance.

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++, Python`, or any other programming language, **as long as it can be easily run on the ECS systems**.

You should submit the following files electronically:

- (15 marks) `Program code` for your k-Nearest Neighbour Classifier (the source code as well as the executable program that runs on the ECS School machines[1]).
- `readme.txt` which describes how to run your program.

[1] If you use Python or another interpreted language, the source and executable programs are the same!

- (15 marks) A report in `.pdf` format. The report should include:

    1. Report the class labels of each instance in the test set predicted by the basic nearest neighbour method (where k=1), and the classification accuracy on the test set of the basic nearest neighbour method. Make sure you keep the same order as in the test set file and remember to divide by the feature range for each feature.
    2. Report the classification accuracy on the test set of the k-nearest neighbour method where k=3, and compare and comment on the performance of the two classifiers (k=1 and k=3).
    3. Discuss the main advantages and disadvantages of k-Nearest Neighbour method.
    4. Assuming that you are asked to apply the k-fold cross validation method for the above problem with k=5, what would you do? State the major steps.
    5. In the above problem, assuming that there were actually no class labels available. Which method would you use to group the examples in the data set? State the major steps.

- This question is *compulsory* for AIML420 students (7 marks). It is *optional* for COMP307 students (who can receive up to 5 bonus marks)

    Implement the clustering method from above and run it on the *wine* data set using the number of clusters (k) of 3 and 5. Submit the program code, report the number of instances in each cluster, and provide an analysis on your results. Simple comments that compare your results with the class labels is sufficient.

Part 2: Decision Tree Learning Method (35 Marks for COMP307, and 38 Marks for AIML420)

This part involves writing a program that implements a simple version of the Decision Tree (DT) learning algorithm, reporting the results, and discussing your findings.

Problem Description

The main data set for the DT program is in the files `hepatitis, hepatitis-training`, and `hepatitis-test`. It describes 137 cases of patients with hepatitis, along with their outcomes. Each case is specified by 16 Boolean attributes, which describe 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 112 of the cases (chosen at random) and the testing file contains the remaining 25 cases. The *first* columns of the files show the class label ("live" or "die") of each instance. The data files are formatted as tab-separated text files, containing one header line, followed by a line for each instance:

- The first 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 (https://archive.ics.uci.edu/ml/datasets/hepatitis). This version has been simplified by removing some numerical attributes, and converting others to Booleans.

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. This 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 relatively simple. Complications arise when handling multiple kinds of attributes, doing statistical significance testing, pruning the tree, etc., but you don't need to deal with these issues in this assignment!

For the simplest case of constructing a decision tree for a set of instances with Boolean attributes (yes/no decisions), with no pruning, the algorithm is shown below. Note that this is a recursive algorithm.

```
instances: the set of training instances that have been provided 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 that contains the name and probability of the most probable
               class across the whole training set (i.e. the ‘‘baseline’’ predictor)
   else if  instances are pure (i.e. all belong to the same class):
      return a leaf node that contains the name of the class and probability 1
   else if  attributes is empty:
      return a leaf node that contains the name and probability of the majority
               class of instances (chosen randomly if classes are equal)
   else find best attribute:
      for each attribute:
         separate instances into two sets:
              1) instances for which the attribute is true,  and
              2) 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 next branch to take based 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 use any programming language, **as long as it can be easily run on the ECS systems**.

You should submit the following files electronically:

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

  You should use the (im)purity measures presented in the lectures unless you want to use another measure from the textbook (e.g. information gain), which is more complex and not recommended. If you choose to do so, please make this clear in your report. The file `helper-code.java` contains java code that helps to read instance data from the data files. You may use it if you find it useful.

- `readme.txt` describing how to run your program.

- (15 marks) A report in `.pdf` format. The report should include:

  1. You should first apply your program to the `hepatitis-training` and `hepatitis-test` files and report the classification accuracy in terms of the fraction of the test instances that it classified correctly. Report the constructed decision tree classifier printed by your program. Compare the accuracy of your decision tree program to the baseline classifier (which always predicts the most frequent class in the training set), and comment on any difference.
  2. You should then apply 10-fold cross-validation to evaluate the robustness of your algorithm. We have provided files for the split training and test sets. The files are named as `hepatitis-training-run-*`, and `hepatitis-test-run-*`. Each training set has 107 instances and each test set has the remaining 30 instances. You should train and test your classifier on each pair, and calculate the average accuracy of the classifiers across the 10 folds (show your working).
  3. “Pruning” (removing) some of leaves of the decision tree will 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 reduces accuracy on the training set; and (c) why it might improve accuracy on the test set.

4. Explain why the impurity measure (from lectures) is not an appropriate measure to use if there are three or more classes in the dataset.

- This question is only for AIML420 students (3 marks). What three conditions must be met if a function (such as $P(A) \times P(B)$) is used as an impurity measure in building a decision tree?

Note: A Simple Way of Outputting a Learned Decision Tree

The easiest way of outputting the tree is to do a (depth-first!) traversal of the tree. For each non-leaf (internal) node, print out the name of the attribute, and 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 probability. 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 (windy = False) is impure, which can only occur on a path that has already used all the attributes, meaning there is no attribute left to split the instances any further. This does not happen on all 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 that may be helpful.

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

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

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

```
public void report(String indent){
 if (probability==0){ //Error-checking
   System.out.printf("%sUnknown%n", indent);
 }else{
   System.out.printf("%sClass %s, prob=%.2f%n", indent, className, probability);
 }}
```

Part 3: Perceptron (35 Marks for COMP307, and 45 Marks for AIML420)

This part of the assignment involves writing a program that implements a **perceptron** that learns to distinguish between two classes of radar data for the *ionosphere* data set.

Data Set

The *ionosphere* data set is taken from the UCI Machine Learning Repository (`https://archive.ics.uci.edu/ml/datasets/ionosphere`). The data set contains 351 instances in 2 classes, of which 225 are “good” (g) radar samples and the remaining 126 are “bad” (b). Each instance has 34 features, consisting of 17 complex number pairs of pulse numbers. These samples were collected by a system in Goose Bay, Labrador. The system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts.

The file `ionosphere.data` consists of the 351 instances, with a header line on the first line of the file.

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 (w_0), and one for each feature. Given an instance with features $f_1, ..., f_n$, the perceptron will classify the instance as a positive instance (i.e. a member of the class) if:

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

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

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

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

Your program should implement this algorithm. It should present the whole sequence of training examples, several times over, until either the perceptron is correct on all the training examples, or it stops converging (e.g. it has presented all the examples 100 times without any progress). It should then use the perceptron to classify all the examples (without changing the weights anymore) and print the final classification accuracy.

Requirements

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

- load the set of instances from the data file;
- construct a perceptron that uses the features as inputs;
- train the perceptron until either it is correct on all instances, or it stops converging (at least 100 iterations);
- report on the number of training iterations to convergence, or the number of instances that are still classified wrongly; and
- print out the final set of weights the perceptron learned.

You can use any programming language, **as long as it can be easily run on the ECS systems**.

In this part of the assignment, you should submit:

- (25 marks) `Program code` for your perceptron (the source code as well as the executable program that runs on the ECS School machines).
- `readme.txt` describing how to run your program.
- (10 Marks) A report in `.pdf` format. The report should:
  1. Report on the accuracy of your perceptron. For example, did it find a correct set of weights? Did its performance change much between different runs?
  2. Explain why evaluating the perceptron's performance on the training data is not a good measure of its effectiveness. For an A+, you should split the dataset and perform a fairer evaluation.
- This question is only for AIM420 students (10 marks). Use a perceptron to solve the classification problem shown in Table 1. Submit the program code and answer the following two questions in the report:
  1. Run the program code of perceptron to solve this problem five times and report the averaged accuracy.
  2. Analyse the results and make your conclusions in the report.

3 Relevant Data Files

The relevant data files, information files about the data sets, and some utility program files can be found as a `.zip` file on the course homepage (under Assignments). In the `.zip` file, there are three subdirectories: `part1, part2`, and `part3`, which correspond to the three parts of the assignment, respectively.

Table 1: Dataset (for AIML420 students)

Instance	F1	F2	F3	Class
1	0	0	1	0
2	0	1	0	1
3	1	0	1	1
4	1	1	0	0
5	1	1	1	0
6	1	0	0	1
7	0	1	1	1
8	0	0	0	0

4 Assessment

We will endeavour to mark your work and return it to you as soon as possible, hopefully in 2 weeks. Please be patient if COVID-19 prevents us marking as promptly as we plan.

The tutors will run a number of help desks to provide guidance (but won't tell you the answers!).

5 Submission Guidelines

5.1 Submission Requirements

1. Programs for all individual parts. To avoid confusion, the programs for each part should be stored in a separate directory `part1/, part2/, ...`. Within each directory, please provide a `readme.txt` file that specifies how to compile and run your programs on the **ECS School machines**. An output file called `sampleoutput.txt` should also be provided to show the output of your program running properly. If your programs cannot run properly, provide a `buglist` file which details what does and doesn't work.

2. The report as a single PDF. You should mark each of the three parts clearly.

5.2 Submission Method

The programs and the report should be submitted through the web submission system from the COMP307 or AIML420 course web site **by the due time**. Please make sure you submit to the course you are enrolled in.

Please check **again** that your programs can be run on the ECS machines easily according to your `readme`. If the tutors can't run your code, you may **lose marks!** Each tutor has a limited amount of time (< 5 minutes) to get your code running, so please don't ask them to use Pycharm, IntelliJ IDEA, Visual Studio, etc to run your code. All these IDEs support exporting runnable code.

5.3 Late Penalties

The assignment must be submitted on time unless you have made a prior arrangement with the **course coordinator** or have a valid medical excuse. This year, we are using the ECS extension system for all extension requests. Please make a request there if you think you have a valid reason.

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.

5.4 Plagiarism

Plagiarism in programming (copying someone else's code) is just as serious as written plagiarism, and is treated accordingly. Make sure you explicitly write down where you got code from (and how much of it) if you use any other resources asides from the course material. Using excessive amounts of others' code may result in the loss of marks, but plagiarism could result in zero marks!

Every year, we catch students plagiarising. Don't be the one!