CSC242 Intro to AI
Project 4: Learning

In this project you will have the opportunity to implement and evaluate one or more machine learning algorithms. This is a huge area of application and research, and is well-covered in upper-level Computer Science courses. It's also nearing the end of the term. So we can really on scratch the surface with this project. But remember, the more you do and the more you do yourself, the more you'll learn.

For the project, you must implement AT LEAST ONE of the following forms of machine learning:

- Decision tree learning (AIMA 18.3)
- Linear classifiers (AIMA 18.6)
- Neural networks (AIMA 18.7)

Each of these is described in more detail below. Whichever you choose to do, you may implement the others for extra credit (max 20% each, max total extra credit 40%). You would learn a lot, but don't kill yourself. It's the end of term and even 40% extra credit is not that many points.

For this project, you must also produce a short report describing what you did and presenting the results of your learning programs. The requirements for this are also detailed below.

Finally, if you're looking for information about machine learning on the Internet, be very careful NOT to look at code. . . especially code from this course. . .

Decision Tree Learning

Decision tree learning is well covered in AIMA Sect. 18.3.

You should think about what it takes to represent a decision tree in a computer program. **THINK ABOUT IT NOW**.

Ok, I hope you thought about trees, whose nodes are either attributes to test (at the internal nodes) or values to return (at the leaves), as well as the attributes and their domains of values themselves. You can easily write classes to represent these things.

You will learn the most if you develop your implementation yourself. If you need a little help getting started, look at the documentation for the code I have provided from package "`dt`". It will suggest some classes and give you their APIs. Perhaps that's enough for you to write the code. But if you need a bit more help, you can build off the code I have provided. You will need to implement some crucial method(s) for actually learning the decision tree.

If you choose this topic, you MUST implement the entropy-based attribute selection method described in Sect. 18.3.4. (You might do something simpler first, but for full points you must do the real calculation, which is the basis of the ID3 algorithm.) You should be able to replicate the textbook results for the restaurant *WillWait* example. Note that for other problems with non-Boolean variables, you will need to use the full definition of entropy and information gain, not just the Boolean case described in the textbook (that is, function H not just function B).

Demonstrate your program on the restaurant example and the Iris dataset ("discrete" version), both of which are provided in our code bundle. Feel free to try other datasets also. See below for details regarding your report.

Linear Classifiers

Linear classifiers are well covered in AIMA Sect. 18.6.

Think about what it takes to represent a linear classifier and the data used to train it in a computer program. **THINK ABOUT IT NOW**.

Ok. I hope you thought about things like input vectors, outputs, weight vectors, and update rules. None of these are hard to implement, but you should think through the design before jumping in with unstructured code.

You will learn the most if you develop your implementation yourself. If you need a little help getting started, look at the documentation for the code I have provided from package "`lc`". It will suggest some classes and give you their APIs. That may be enough for you to write the code. But if you need a bit more help, you can build off the code I have provided. You will need to implement the crucial classes and/or methods for actually learning the linear classifiers.

If you choose this topic, you MUST implement both a perceptron classifier (18.6.3) and a logistic classifier (18.6.4). Almost all of the code can be shared if you design it right. You should be able to replicate something like the textbook results for the earthquake problem (Figures 18.15, 18.16, and 18.18).

Demonstrate your program on the earthquake data (both clean and noisy datasets) and the "house votes" dataset ("numerical" version), all of which are provided in our code bundle. Feel free to try other datasets also. See below for details regarding your report.

Neural Networks

Neural networks are covered in AIMA Section 18.7. It does cover all the important definitions for both single-layer and multi-layer feed-forward networks, and it provides the algorithm for backpropagation in multi-layer networks (Fig. 18.24). That said, it is very concise. So if you choose to implement this type of learner, be prepared to do some thinking and/or additional research as you develop and evaluate your system.

It isn't hard to think about what you need to represent a neural network in a computer program. **THINK ABOUT IT NOW**.

Ok. I hope you thought about "units," layers, connections, weights, activation functions, inputs, and outputs. Remember that a neural network is simply a graph of linear classifiers (typically using a logistic threshold). It is not hard to design classes incorporating these elements. However I suggest that you understand how the backpropagation algorithm works before you lock in your design. In particular, note that it requires that you be able to go both forward and backward through the layers of your networks, even though the network is "feed-forward."

As always, you will learn the most if you develop your implementation yourself. And in this case, if you need a little help, I'm sorry but the code I can provide is less useful. YLike the other two types of learning systems, you are welcome to look at the documentation for the package "`nn`". That will give you some suggestions for classes and APIs. And if you need more help, you can build off the code I have provided. You will need to implement the crucial classes and/or methods for actually learning the linear classifiers in `nn.core`.

If you choose this topic, you MUST implement a multi-layer network with hidden units. A single-layer network is essentially a set of one or more linear classifiers. A multi-layer network is a "true" neural network that must be trained using backpropagation (AIMA Fig. 18.24).

Demonstrate your program on the Iris dataset and the MNIST dataset (handwritten digit recognition). Files, or pointers to data file in the case of MNIST, are provided in our code bundle. For the Iris dataset, try a two-layer network with four inputs, seven hidden units, and three output units (one for each species of Iris). For MNIST, visit `http://yann.lecun.com/exdb/mnist/` and try some of the networks described there. For example, a two-layer network with 300 or 1000 hidden units, or a three-layer network with 500 and 150 units (first and second layer, respectively). As the textbook says: "Unfortunately, for any *particular* network structure, it is harder to characterize exactly which functions can be represented and which ones cannot."

Feel free to try other datasets also. See below for details regarding your report.

Datasets for Machine Learning

There are many, many datasets available online for training different types of machine learning systems. One of the best sources is the UCI Machine Learning Archive: `http://archive.ics.uci.edu/ml`. I have included some datasets from this archive with the sample code for this project.

There is always some work reading these files and getting them into the proper form for your learning system. It's easy for simple datasets, like the "Iris" dataset, and harder for more complicated datasets. For problems based on images (or other media), there is often a dataset with a set of attribute values extracted from the data already available, so that you can focus on machine learning rather than image processing. Or you may be interested in extracting the attributes from the media yourself. Note that continuous-valued datasets may need to be discretized for some types of learning (unless you want to implement more sophisticated algorithms).

Start simple. Use the restaurant example for decision trees, the earthquake dataset for linear classifiers, or the Iris dataset for neural networks to get started. They are manageable. Then try something bigger and more interesting.

Report Requirements

Your report must include the exact commands needed to run your programs. You may assume that the working directory is the parent of the `src` directory.

If you are using Eclipse and don't understand that, make sure that there are Run Configurations setup for each way the programs should be run and tell us in your report *exactly* what to do to run them.

Either way, it is your job to make this clear to us.

Decision Trees

Your programs' output must include printing the learned decision tree in a clear and understandable way (I recommended some kind of indented output).

For the restaurant *WillWait* problem, there are only twelve training examples, so have your program test on the training data and output the accuracy. You should know what to expect. Summarize the results in your report.

For the Iris database, there are 150 examples. You should do k-fold cross-validation with $k = 5$ and output the individual accuracies and the average over the 5 runs. Summarize the results in your report.

You are welcome to try additional problems or evaluation methods and include them in your report.

Linear Classifiers

Your programs' output must include data sufficient to produce *training curves* as seen in AIMA Fig. 18.16 and 18.18. And then you must use that output to produce the graphs and include them in your report (clearly labelled and explained). You may use Excel or Google Docs or `gnuplot` or make your own plotter for this.

Your report should include the commands necessary to produce the data for all six graphs from AIMA for the earthquake data, and include the graphs themselves. For the "house votes" dataset, do something similar and include the commands and the graphs in your report.

Neural Networks

For each problem, your program's output should include the overall accuracy (on the training data) after training for some number of epochs, for example $N = 1000$.

You should also perform k-fold cross-validation for some reasonable k such as $k = 10$ and output the results of each trial and the overall average accuracy.

And finally your programs should try training for a varying number of epochs from 100 to 3000 by 100s (or something else reasonable) and output data sufficent to produce curves similar to those shown in AIMA Fig. 18.25. Measure accuracy on the training data and plot that against number of epochs of training to get a curve that is the inverse of Fig 18.25 (a) (which shows error vs number of eopchs). You could also plot error, which is $1.0 - accuracy$.

You should include all these results including the curves in your report.

Programming Practice

Use good object-oriented design. No giant `main` methods or other unstructured chunks of code. Comment your code liberally and clearly.

You may use Java, Python, or C/C++ for this project. I recommend that you use Java. Any sample code we distribute will be in Java. Other languages (Haskell, Clojure, Lisp, *etc.*) by arrangement with the TAs only.

You may **not** use any non-standard libraries. Python users: that includes things like NumPy. Write your own code—you'll learn more that way.

If you use Eclipse, make it clear how to run your program(s). Setup Build and Run configurations as necessary to make this easy for us. Eclipse projects with poor configuration or inadequate instructions will receive a poor grade.

Python projects must use Python 3 (recent version, like 3.6.x). Mac users should note that Apple ships version 2.7 with their machines so you will need to do something different.

If you are using C or C++, you should use reasonable "object-oriented" design not a mish-mash of giant functions. If you need a refresher on this, check out the C for Java Programmers guide and tutorial. You **must** use "`-std=c99 -Wall -Werror`" **and**

have a clean report from `valgrind`. Projects that do not follow both of these guidelines will receive a poor grade.

Late Policy

Late projects will **not** be accepted. Submit what you have by the deadline. If there are extenuating circumstances, submit what you have before the deadline and then explain yourself via email.

If you have a medical excuse (see the course syllabus), submit what you have and explain yourself as soon as you are able.

Collaboration Policy

You will get the most out of this project if you write the code yourself.

That said, collaboration on the coding portion of projects is permitted, subject to the following requirements:

- Groups of no more than 3 students, all currently taking CSC242.
- You must be able to explain anything you or your group submit, IN PERSON AT ANY TIME, at the instructor's or TA's discretion.
- One member of the group should submit code on the group's behalf in addition to their writeup. Other group members should submit only a README indicating who their collaborators are.
- All members of a collaborative group will get the same grade on the project.

Academic Honesty

Do not copy code from other students or from the Internet.

Avoid Github and StackOverflow completely for the duration of this course.

There is code out there for all these projects. You know it. We know it.

Posting homework and project solutions to public repositories on sites like GitHub is a violation of the University's Academic Honesty Policy, Section V.B.2 "Giving Unauthorized Aid." Honestly, no prospective employer wants to see your coursework. Make a great project outside of class and share that instead to show off your chops.