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, undergraduates must implement AT LEAST ONE of the following forms of machine learning, and graduate students must implement AT LEAST TWO of them. The choices are:

- 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 10% each, max total extra credit 20%).

Your writeup is important for this project. You need to convince us that your implementation works AND that you know how to evaluate a machine learning program. We should be able to read your writeup, then run the program(s) and compare the output with the results in your writeup, then look at your code to see how you did it. IT IS UP TO YOU TO MAKE THIS CLEAR.

If you're looking for information 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, we 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. However, if you need a little help getting started, look at the documentation for the code we 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 we 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 may 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 at least one other problem of your choosing (for example, the discrete Iris dataset provided in our code bundle).

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. We hope you thought about things like inputs, 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. However, if you need a little help getting started, look at the documentation for the code we 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 we 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 are provided in our code bundle) and at least one other problem of your choosing (for example, the "house votes" dataset provided in our code bundle).

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. We hope you thought about "units," layers, connections, weights, activation functions, inputs, and outputs. 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 we can provide is less useful. You are welcome to look at the documentation for the package "`nn`". That will give you some suggestions for classes and APIs. But there are gaps in our implementation that you will have to fill in.

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). Our code bundle includes examples of testing a single-layer network on the "XOR" problem described in AIMA, as well as multi-layer examples for the Iris dataset and the "MNIST" handwritten-digit classification problem.

You need to demonstrate your program on at least one reasonable dataset, and doing two would be better. The textbook claims to use a set of 100 restaurant examples to produce Fig 18.25, but unfortunately, we don't have that dataset. We found a reference claiming that the UCI Iris dataset could be learned by a network with four inputs, seven hidden units, and 3 output units. So you could try that, or try another problem. 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." In other words, designing neural networks is something of an art.

Whatever you do, document it clearly (what and why) in your writeup and include graphs as necessary to support your description.

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`. We 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. Trying is not the same as getting it done, but it counts for something. Whatever you do: explain clearly in your writeup what you did and why, and provide some evaluation like that shown in the textbook.

Evaluation of Machine Learning Systems

Machine learning algorithms and their implementations generally have many parameters that can affect their speed and quality. This is arguably more true for machine learning than for the previous projects in this course. It is therefore important that you think about **and evaluate and report on** these factors in the context of your program(s).

At the very least, your writeup MUST have some graphs in it. The textbook has examples of different types of learning curves. We have tried to point out these out in the descriptions of the problems given above. The important thing is that a single run of a learning algorithm is almost never informative. If you are uncertain about how to report your work, come to one of the study hall sessions and discuss it with us.

For your consideration, here is a short, non-exhaustive list of the sorts of parameters typically found in some or all approaches to machine learning.

- Loss function
- Learning rate (α)
- Entropy function
- Relative size of training vs. testing sets
- Number of partitions for cross-validation (k)
- Initialization of weights and bias terms
- Batch size for stochastic/batch/minibatch
- λ for regularization

Most of these are covered in AIMA and you should be able to do experiments and comment on at least some of them in your writeup. Any questions, bring them to a study session.

About Your Programs

We must be able to build your code per the instructions in your README or writeup. As always in this course, we generally use the latest Fedora Linux with the latest gcc, java, or python. The project description section “Programming Practice” describes the requirements in more detail.

We must be able to run your programs per the instructions in your writeup.

These instructions must demonstrate both the training of the model AND the use of the trained model. Typically you do these together anyway, for example to generate statistics for your learning curves and other graphs.

If the training phase takes hours or days, you have two options:

1. You can write the trained model to a file, and have a program that loads a model from a file and runs it on a set of examples. Then include trained models that took a long time to train with your submission, with clear instructions about how to produce and how to use them.
2. Give us a faster but perhaps less accurate version that we can run in a reasonable amount of time. Of course, you can still include the results of better, slower versions in your writeup.

Make it clear in your writeup how to run your program in these different modes (without editing the code: either different programs using shared code or different command-line options to a single program).

Bottom line: We want to be able to build your program(s), run them in “training” mode on some data to learn a model, and run them in “testing” mode on some other data.

Your writeup must make it clear how to do this in a reasonable amount of time. Your programs must produce clear, meaningful output and clear, useful statistics. Your writeup must summarize and present the results as we have seen in the textbook and in class.

Any questions, bring them to a study session.

Project Submission

Your project submission MUST include the following:

1. A README.txt file or PDF document describing:
   (a) Any collaborators (see below)
   (b) How to build your project
   (c) How to run your project's program(s) to demonstrate that it/they meet the requirements
2. All source code and build files for your project (see below)
3. A writeup describing your work in PDF format (see below)

We must be able to cut-and-paste from your documentation in order to build and run your code. **The easier you make this for us, the better grade your will be.** It is your job to make both the building and the running of programs easy and informative for your users.

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, you *must* make it so that we can build and run your project without Eclipse. Document exactly what needs to be done in your README (Makefile, `javac` or `gcc` incantations, whatever). Eclipse projects with no build instructions will receive 0.

Your code must build on Fedora Linux using recent versions of Java, Python, or `gcc`.

- This is not generally a problem for Java projects, but don't just submit an Eclipse project without the build and run instructions detailed above.
- 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 must use "`-std=c99 -Wall -Werror`" and have a clean report from `valgrind`. And you'd better test on Fedora Linux or expect problems.

Writing Up Your Work

As noted above, it is crucial that you present your work clearly, honestly, and in its best light. We will give lots of credit for good writeups. We will not like submissions with sloppy, unstructured, hard to read writeups that were clearly written at the last minute.

Your goal is to produce a technical report of your efforts for an expert reader. You need to convince the reader that you knew what you were doing and that you did it as best you could in the (entire) time available.

Write up what you did (and why). If you didn't get to something in your code, write about what you might have done. (There's always *something* you might have done.) If something didn't work, write about why and what you would do differently. But don't write "I would have started sooner." Your readers already know that.

Your report *must be your own words*. Material taken from other sources must be properly attributed if it is not digested and reformulated in your own words. **Plagiarism is cheating.**

Start your writeup early, at least in outline form. Document your design decisions as you make them. That way half the write-up is done by the time you're running the program. Don't forget to include illustrations of the program in action (traces, screenshots) and some kind of evaluation.

Your report must be typeset (not handwritten). Using LaTeX and BibTeX makes things look nice and is worth learning anyway if you don't know it, but it's not a requirement. Your report must be submitted as a PDF file. If you want to use a word processor, be sure to generate and submit a PDF from it, not the document file.

The length of the report is up to you. For a CSC242 assignment, I would say that 3–5 pages is the *minimum* that would allow you to convince the reader. Be sure to include screenshots and/or traces (which wouldn't count towards the 3-5 pages minimum of actual text).

Late Policy

Don't be late. But if you are: 5% penalty for the first hour or part thereof, 10% penalty per hour or part thereof after the first.

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 their writeup.
- All members of a collaborative group will get the same grade on the coding component of the project.

You may NOT collaborate on your writeup. Your writeup must be your own work. Attribute any material that is not yours. Cite any references used in your writeup. Plagiarism is cheating.