Why?

The purpose of this project is to explore some techniques in supervised learning. It is important to realize that understanding an algorithm or technique requires understanding how it behaves under a variety of circumstances. As such, you will be asked to "implement" some simple learning algorithms (for sufficiently small values of implement, meaning I don't really want you to implement anything at all), and to compare their performance.

You may program in any language that you wish (we do prefer broadly-used or revered languages like java, matlab, python, Lisp, R, or even C++; let us know beforehand if you're going to use something else). In any case *it is your responsibility* to make sure that the code can run on the standard CoC linux boxes. Re-read that last sentence. Next, ask yourself what it could possibly mean in practice. Well, in practice, it means that if a TA needs to recreate your experiments, it will be possible (as detailed below) and if it isn't possible because you used some obscure language you made up based on HTML 2.0, well, that's just not going to work out. Some of you who have been out of school for a long time and/or are used to assignments where you are told exactly what to do will fret about this requirement and try to infer more than is there. Don't. Use standard packages or languages and you'll never have to even figure out how to log into any CoC machine.

Read everything below carefully!

The Problems Given to You

You should implement five learning algorithms. They are for:

- Decision trees with some form of pruning
- Neural networks
- Boosting
- Support Vector Machines
- *k*-nearest neighbors

Each algorithm is described in detail in your textbook, the handouts, and all over the web. In fact, instead of implementing the algorithms yourself, you may (and by may I mean should) use software packages that you find elsewhere; however, if you do so you should provide proper attribution. Also, you will note that you have to do some fiddling to get good results, graphs and such, so even if you use another's package, you may need to be able to modify it in various ways.

Decision Trees. For the decision tree, you should implement or steal a decision tree algorithm (and by "implement or steal" I mean "steal"). Be sure to use some form of pruning. You are not required to use information gain (for example, there is something called the GINI index that is sometimes used) to split attributes, but you should describe whatever it is that you do use.

Neural Networks. For the neural network you should ~~implement or~~ steal your favorite kind of network and training algorithm. You may use networks of nodes with as many layers as you like and any activation function you see fit.

Boosting. Implement===steal a boosted version of your decision trees. As before, you will want to use some form of pruning, but presumably because you're using boosting you can afford to be much more aggressive about your pruning.

Support Vector Machines. You should implement (for sufficently loose definitions of implement including "download") SVMs. This should be done in such a way that you can swap out kernel functions. I'd like to see at least two.

***k*-Nearest Neighbors.** You should "implement" *k*NN. Use different values of *k*.

Testing. In addition to implementing (wink), the algorithms described above, you should design two interesting classification problems. For the purposes of this assignment, a classification problem is just a set of training examples and a set of test examples. I don't care where you get the data. You can download some, take some from your own research, or make some up on your own. Be careful about the data you choose, though. You'll have to explain why they are interesting, use them in later assignments, and come to really care about them.

What to Turn In

You must submit via tsquare a tar or zip file named *yourgtaccount*.{zip,tar,tar.gz} that contains a single folder or directory named *yourgtaccount* (where *gtaccount* is probably something like *gtg7577b* or maybe *ci16*). That directory in turn contains:

1. a file named *README.txt* containing instructions for running your code (see note below)
2. your code (see note below)
3. a file named *yourgtaccount-analysis.pdf* containing your writeup
4. any supporting files you need, such as your training and test sets (see note below).

Note below: if the data are way, way, too huge for submitting, see if you can arrange for an URL. This also goes for code, too. Submitting all of Weka isn't necessary, for example, because I can get it myself; however, you should at least submit any files you found necessary to change and enough support and explanation so we could reproduce your results if we really wanted to do so. In any case, include all the information in *README.txt*

The file *yourgtaccount-analysis.pdf* should contain:

- a description of your classification problems, and why you feel that they are interesting. Think hard about this. To be at all interesting the problems should be non-trivial on the one hand, but capable of admitting comparisons and analysis of the various algorithms on the other.

- the training and testing error rates you obtained running the various learning algorithms on your problems. At the very least you should include graphs that show performance on both training and test data as a function of training size (note that this implies that you need to design a classification problem that has more than a trivial amount of data) and--for the algorithms that are iterative--training times.

- analyses of your results. Why did you get the results you did? Compare and contrast the different algorithms. What sort of changes might you make to each of those algorithms to improve performance? How fast were they in terms of wall clock time? Iterations? Would cross validation help (and if it would, why didn't you implement it?)? How much performance was due to the problems you chose? How about the values you chose for learning rates, stopping criteria, pruning methods, and so forth (and why doesn't your analysis show results for the different values you chose?)? Which algorithm performed best? How do you define best? Be creative and think of as many questions you can, and as many answers as you can.

For the sanity of your graders, please keep your analysis as short as possible while still covering the requirements of the assignment: to facilitate this sanity, **analysis writeup is limited to 12 pages.**

Grading Criteria

You are being graded on your analysis more than anything else. Roughly speaking, implementing everything and getting it to run is worth maybe 0% of the grade on this assignment (I know you don't believe me, but in fact, steal the code; I not only don't care, I am encouraging you to use one of the many packages available both from the resources page and on the web). Of course, analysis without proof of working code makes the analysis suspect.

The key thing is that your explanations should be both thorough and concise. Imagine you are writing a paper for the major conference in your field the year you will be graduating and you need to impress all those folks who will be deciding whether to interview you later. You don't want them to think you're shallow do you? Or that you're incapable of coming up with interesting classification problems, right? And you surely don't want them to think that you make up for a lack of content by blathering on about irrelevant aspects of your work? Of course not.

Finally, I'd like to point out that I am very particular about the format of the assignments. Follow the directions carefully. Failure to turn in files without the proper naming scheme, or anything else that makes the graders' lives unduly hard is simply going to lead to an ignored assignment. I am remarkably inflexible about this. Also, there will be no late assignments accepted, so start now. Have fun. One day you'll look back on this and smile. There may be tears, but they will be tears of joy.