CSC 425/525 Programming Assignment #4
Due Date: Wednesday, April 26

Choose one of the following learning algorithms to implement. Some of these require that you implement the learning algorithm from scratch. If one allows you to use a library, then your grade will be more based on the results of your system rather than the implementation. You will select your own domain and the specific learning problem within that domain (although some suggestions are listed). You may also propose your own learning algorithm for consideration.

1. FF/BP #1 – build your own neural network program which implements both FF and BP in a 3-layered network (1 hidden layer). Then train a neural network on training data and test it and see how it does. In your report, talk about training time and testing results.

2. FF/BP #2 – rather than implementing your own neural network, use a library that will let you specify the size/shape of the network and then train it on training data. When done, test it on testing data and see how it performed. Repeat this multiple times with different sizes/shapes and with different training/testing data. Your goal is to, for the given domain, identify the best neural network based on two things: how quickly it converged and how accurate it is with the testing data.

3. SVMs – use a library to train several SVMs, one per category in the class. Select training data to train the SVMs and then testing data to test them. See how accurate your results are. If inaccurate, try two different things. First, try different combinations of data to train and then test. If this doesn't improve the accuracy, try a different kernel. Keep training SVMs until you have a reasonably good accuracy.

For the above, a good domain to try is hand-written character recognition where the input is a bitmap of a single character. You can either create your own bitmaps using a tool like paint, or you can write characters and scan them in. For #1, you might limit the recognition task to just 2-4 total characters like digits 0, 1 or 2. For #2 and #3, use a decent sized number of characters such as the 10 digits or 26 upper case letters.

4. Implement a genetic algorithm to solve either an optimization problem or a planning/creation problem. For instance, you might try to solve the bin packing problem or knapsack problem, or you might write a program that can generate haikus or other forms of poetry. For poetry, have a database of words to select from, and make a chromosome be the entire haiku, or one line in the poem. For a haiku, the fitness function would have to consider such things as number of syllables per line and whether the words "flow". For a poem, the fitness function should consider the types of grammatical categories of the words (is it grammatically correct) and whether the words relate to each other. Another example is to use a GA to generate a plan for your daily activities where a fitness function critiques such things as "if you go to the store and then go somewhere without returning home, the groceries may spoil".

5. Implement either a genetic programming system or an evolutionary programming system as follows. Write a system which can take a program's source code as input, make random changes to the program through genetic operations, evaluate the changes through a fitness function and select the best one(s) to become a parent for a new generation. The fitness function will first require that the child programs be compiled and then run on a collection of test data and evaluate the number of correct and incorrect outputs. Genetic operations can change the order of instructions, mutate assignment statements within a restricted

amount (you can change + to *, etc or change a variable to another variable of the same type) and you can cross-over same-typed instructions (e.g., two arithmetic expressions). Write a few partially functioning programs (ones that have the right set of instructions but perhaps not in the right order or with some instructions being incorrect) such that when run on the test data, they get a few of the test data correct but not a majority. See if your GP/EP system can evolve a correct program. Find cases where it works well and cases where it doesn't work well or at all. NOTE: you might want to write your program code in an interpreted language like Python so that you can directly execute a textfile source program without having to compile it. As an example program, you might write a program that is supposed to perform sequential search but is slightly erroneous (logically) and see if you can evolve a working version.

6. Use a library to implement the C4.5 decision tree learning algorithm. Find a dataset that consists of at least several hundred records and select a field that you want to learn where there are between 5 and 10 different possible outcomes for the given field (for instance, identifying a major given a student's interests). Do not make up your own data! Build your decision tree and then test it out. As time permits, remove some of the data and build a new tree. Try to find a reasonable subset of data by which you can build a useful decision tree for the domain you selected.

7. Implement a naïve Bayesian classifier that uses the "bag of words" approach to learn to classify text documents into one of several different classes. For instance, build 4 NBCs, one each for 4 different classes (e.g., computer science documents, philosophy documents, psychology documents, neurology documents). This will require that you save several dozen documents for each class. Split the documents into training documents and testing documents. Write the NBC code, train your NBCs, and then test each on separate documents. If your results are not good, go back to your NBC training algorithm and try to improve on it. For instance, you might initially try to train the NBCs without weighing word occurrences and then use weights, or train on individual words and if that doesn't work, train on phrases.

8. Use a library to implement Bayesian network in some domain for a diagnostic problem that you are comfortable with. You will have to accumulate appropriate statistical data to build the network but make sure it is a domain that you have enough knowledge of to work with the software to build the model. Use 3/4s of the data to build the system and the remaining data to test the system. If your tests are not accurate, rebuild the system using a different subset of test data.

Hand in:

1) the listing of your program and a description of the training data and some example runs.
2) a report describing what learning algorithm you selected, how you implemented it (or what library you used), the problem and domain chosen, the number and types of training examples, what your system learned, how long it took, and how well it performed, and what you learned during the implementation.

Undergraduates are allowed to work in groups of up to 3, graduate students must work alone.