CS211 Spring 2018
Programming Assignment II

David Menendez

Due: March 5, 2018, at 5:00 PM

This assignment is designed to give you more experience programming in C and using the Unix environment. Your task will be to write one program that implements a simple machine-learning algorithm. This will require file I/O, dynamic memory allocation, and correctly implementing an moderately complex algorithm.

Machine learning (ML) techniques are increasingly used to provide services, such as face recognition in photographs, spelling correction, automated translation, and predicting what YouTube videos you might want to watch next. Implementing a full ML algorithm is beyond the scope of this course, so you will implement a "one shot" learning algorithm that uses historical data to predict house prices based on particular attributes.

For example, a house might have x_1 bedrooms, x_2 bathrooms, x_3 square footage, and be built in year x_4. If we had appropriate weights, we could estimate the price of the house y with the formula

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4. \tag{1}$$

The goal of one-shot learning is to find values for the weights w_i using a large provided set of training data. Once those weights have been found, they can be used to estimate prices for additional houses.

For example, if the training data includes n houses and has k attributes, this data can be represented as an $n \times (k+1)$ matrix X, of the form

$$\begin{pmatrix} 1 & x_{0,1} & x_{0,2} & \cdots & x_{0,k} \\ 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1,1} & x_{n-1,2} & \cdots & x_{n-1,k} \end{pmatrix},$$

where each row corresponds to a house and each column corresponds to an attribute. Note that the first column contains 1 for all rows: this corresponds to the weight w_0.

Similarly, house prices can be represented as an $n \times 1$ matrix Y, of the form

$$\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix},$$

where each row gives the price of a house.

Finally, the weights will be a $(k+1) \times 1$ matrix W, of the form

$$\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{k+1} \end{pmatrix},$$

where each row gives the weight of an attribute.

The goal of one-shot learning is to find W, given X and Y from the training data.

In matrix notation, we would write

$$XW = Y. \tag{2}$$

If X were a square matrix, we could find W by rewriting the equation as $W = X^{-1}Y$, but in general X will not be a square matrix. Thus, we will find its *pseudo-inverse*, and calculate

$$W = (X^T X)^{-1} X^T Y, \tag{3}$$

where X^T is the *transpose* of X. $X^T X$ is a square matrix, and can be inverted.[1]

Once W has been found, it can be used with a new set of house attributes X' to estimate prices for those houses by computing $X'W = Y'$.

1 Algorithm

Given matrices X and Y, your program will compute $(X^T X)^{-}1 X^T Y$ in order to learn W. This will require (1) multiplying, (2) transposing, and (3) inverting matrices. Programming Assignment I already involved matrix multiplication; you may adapt your implementation for this assignment.

Transposing an $m \times n$ matrix produces an $n \times m$ matrix. Each row of the X becomes a column of X^T.

To find the inverse of $X^T X$, you will use a simplified form of Gauss-Jordan elimination.

1.1 Gauss-Jordan elimination for finding inverses

Gauss-Jordan is a method for solving systems of equations by manipulating matrices with *row operations*. This method can also be used to find the inverse of a matrix.

You will implement two of the three row operations in your program. The first multiplies all elements of a particular row by some number. The second adds the contents of one row to another, element-wise. More generally, the second operation adds a multiple of the elements of one row to another so that element $x_{i,k}$ will become $x_{i,k} + ax_{j,k}$.

The third row operation, which swaps two rows, will not be needed for this assignment. Again, the training data used to grade this assignment will not require swapping rows.

Here is a demonstration of Gauss-Jordan elimination. We begin with the matrix we wish to invert:

$$M = \begin{bmatrix} 1 & 2 & 4 \\ 1 & 6 & 7 \\ 1 & 3 & 2 \end{bmatrix}.$$

[1] This is not true in general, but for this assignment you may assume that $X^T X$ is invertable.

We create an *augmented* matrix $A = M|I$ by adjoining the identity matrix I to M.

$$A = \left[\begin{array}{ccc|ccc} 1 & 2 & 4 & 1 & 0 & 0 \\ 1 & 6 & 7 & 0 & 1 & 0 \\ 1 & 3 & 2 & 0 & 0 & 1 \end{array}\right].$$

We will then apply row operations to A in order to turn its left half into the identity matrix.

We will write A_i to refer to row i of A. At each step of the algorithm, we will identify a particular row as the *pivot row*. The element in the pivot row that lies on the diagonal (that is, element $A_{p,p}$) is the *pivot element*.

The first step is to turn the matrix into an upper triangular matrix, where all elements on the diagonal are 1 and elements below the diagonal are 0. The pivot row will start at A_0 and advance to A_2. At each step, we will first multiply the pivot row by a constant so that the pivot element will become 1. Next, we will subtract the pivot row from the rows below it, so that the elements below the pivot element become 0.

Starting with row A_0, we see that $A_{0,0}$ is already 1. To make the elements below $A_{0,0}$ become 0, we subtract A_0 from A_1 and A_2, yielding

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 4 & 1 & 0 & 0 \\ 0 & 4 & 3 & -1 & 1 & 0 \\ 0 & 1 & -2 & -1 & 0 & 1 \end{array}\right].$$

Next, for pivot A_1 we see that $A_{1,1} = 4$. We divide A_1 by 4 (that is, multiply A_4 by $\frac{1}{4}$).

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 4 & 1 & 0 & 0 \\ 0 & 1 & \frac{3}{4} & -\frac{1}{4} & \frac{1}{4} & 0 \\ 0 & 1 & -2 & -1 & 0 & 1 \end{array}\right].$$

Then, we subtract A_1 from A_2, yielding

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 4 & 1 & 0 & 0 \\ 0 & 1 & \frac{3}{4} & -\frac{1}{4} & \frac{1}{4} & 0 \\ 0 & 0 & -\frac{11}{4} & -\frac{3}{4} & -\frac{1}{4} & 1 \end{array}\right].$$

Next, we divide A_3 by $-\frac{11}{4}$, yielding

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 4 & 1 & 0 & 0 \\ 0 & 1 & \frac{3}{4} & -\frac{1}{4} & \frac{1}{4} & 0 \\ 0 & 0 & 1 & \frac{3}{11} & \frac{1}{11} & -\frac{4}{11} \end{array}\right].$$

A is now an upper triangular matrix. To turn the left side of A into an identity matrix, we will reverse the process and turn the elements above the diagonal into 0. The pivot row will start at A_2 and advance in reverse to A_0. For each step, we will subtract the pivot row from the rows above it so that the elements above the pivot element become 0.

We begin with A_2. First, we subtract $\frac{3}{4}A_2$ from A_1, yielding

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 4 & 1 & 0 & 0 \\ 0 & 1 & 0 & -\frac{5}{11} & \frac{2}{11} & \frac{3}{11} \\ 0 & 0 & 1 & \frac{3}{11} & \frac{1}{11} & -\frac{4}{11} \end{array}\right].$$

Then we subtract $4A_2$ from A_0, yielding

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 0 & -\frac{1}{11} & -\frac{4}{11} & \frac{16}{11} \\ 0 & 1 & 0 & -\frac{5}{11} & \frac{2}{11} & \frac{3}{11} \\ 0 & 0 & 1 & \frac{3}{11} & \frac{1}{11} & -\frac{4}{11} \end{array}\right].$$

Now the elements above $A_{2,2}$.

Next, we subtract $2A_1$ from A_0, yielding

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & \frac{9}{11} & -\frac{8}{11} & \frac{10}{11} \\ 0 & 1 & 0 & -\frac{5}{11} & \frac{2}{11} & \frac{3}{11} \\ 0 & 0 & 1 & \frac{3}{11} & \frac{1}{11} & -\frac{4}{11} \end{array}\right].$$

Now the element above $A_{1,1}$ is 0.

After that step, the left half of A is the identity matrix and the right half of A contains M^{-1}. That is, $A = I|M^{-1}$. The algorithm is complete and the inverse of M has been found.

You are strongly encouraged to write this algorithm in pseudocode before you begin implementing it in C. Try using it to invert a small square matrix and make sure you understand the operations you must perform at each step.

In particular, ask yourself (1) given a matrix A, how can I multiply (or divide) A_i by a constant c, and (2) given a matrix A, how can I add (or subtract) cA_i to (or from) A_j?

Note that the augmented matrix is a notational convenience. In your implementation, you may prefer to use two matrices A and B that are initially equal to M and I. Any time you apply a row operation to A, apply the same one to B. Once you have $A = I$, you will also have $B = M^{-1}$. It is also acceptible to create and manipulate an augmented matrix and to extract M^{-1} from it later.

You MUST use the algorithm demonstrated above. Performing different row operations, or the same row operations in a different order, may change the result of your program due to rounding. This may cause your program to produce results different from the reference result.

2 Program

You will write a program `learn` that uses a training data set to learn weights for a set of house attributes, and then applies those weights to a set of input data to calculate prices for those houses. `learn` takes two arguments, which are the paths to files containing the training data and input data.

Training data format The first line will be the word "train". The second line will contain an integer k, giving the number of attributes. The third line will contain an integer n, giving the number of houses. The next n lines will contain $k + 1$ floating-point numbers, separated by spaces. Each line gives data for a house. The first k numbers give the values $x_1 \cdots x_k$ for that house, and the last number gives its price y.

For example, a file `train.txt` might contain:

```
train
4
7
3.000000 1.000000 1180.000000 1955.000000 221900.000000
```

```
3.000000 2.250000 2570.000000 1951.000000 538000.000000
2.000000 1.000000 770.000000 1933.000000 180000.000000
4.000000 3.000000 1960.000000 1965.000000 604000.000000
3.000000 2.000000 1680.000000 1987.000000 510000.000000
4.000000 4.500000 5420.000000 2001.000000 1230000.000000
3.000000 2.250000 1715.000000 1995.000000 257500.000000
```

This file contains data for 7 houses, with 4 attributes and a price for each house. The corresponding matrix X will be 7×5 and Y will be 7×1. (Recall that column 0 of X is all ones.)

Input data format The first line will be the word "data". The second line will be an integer k, giving the number of attributes. The third line will be an ineteger m, giving the number of houses. The next m lines will contain k floating-point numbers, separated by spaces. Each line gives data for a house, not including its price.

For example, a file `data.txt` might contain:

```
data
4
2
3.000000 2.500000 3560.000000 1965.000000
2.000000 1.000000 1160.000000 1942.000000
```

This contains data for 2 houses, with 4 attributes for each house. The corresponding matrix X will be 2×5.

Output format Your program should output the prices computed for each house in the input data using the weights derived from the training data. Each house price will be printed on a line, rounded to the nearest integer.

To print a floating-point number rounded to the nearest integer, use the formatting code `%.0f`, as in:

```
printf("%.0f\n", price);
```

Usage Assuming the files `train.txt` and `data.txt` exist in the same directory as `learn`:

```
$ ./learn train.txt data.txt
737861
203060
```

Implementation notes You MUST use `double` to represent the attributes, weights, and prices. Using `float` may result in incorrect results due to rounding. To read `double` values from the training and input data files, you can use `fscanf` with the format code `%lf`.

If `learn` successfully completes, it MUST return exit code 0.

You MAY assume that the training and input data files are correctly formatted. You MAY assume that the first argument is a training data file and that the second argument is an input data file. However, checking that the training data file begins with "train" and that the input data file begins with "data" may be helpful if you accidentally give the wrong arguments to `learn` while you

are testing it. To read a string containing up to 5 non-space characters, you can use the `fscanf` format code `%5s`.

`learn` SHOULD check that the training and input data files specify the same value for k.

If the training or input files do not exist, are not readable, are incorrectly formatted, or specify different values of k, `learn` MAY print "error" and return exit code 1. Your code will not be tested with these scenarios.

3 Submission

Your solution to the assignment will be submitted through Sakai. You will submit a Tar archive file containing the source code and makefile for your project. Your archive should not include any compiled code or object files.

The remainder of this section describes the directory structure, the requirements for your makefile, how to create the archive, and how to use the provided auto-grader.

3.1 Directory structure

Your project should be stored in a directory named `src`. This directory will contain (1) a makefile, and (2) any source files needed to compile `learn`. Typically, you will provide a single C file named `learn.c`.

You MAY include a subdirectory `test`, containing additional test files, as described in Section 3.5.

This diagram shows the layout of a typical project:

```
src
+- Makefile
+- learn.c
```

If you are using the auto-grader to check your program, it is easiest to create the `src` directory inside the `pa2` directory created when unpacking the auto-grader archive (see Section 3.4).

3.2 Makefiles

We will use `make` to manage compilation. Each program directory will contain a file named `Makefile` that describes at least two targets. The first target (invoked when calling `make` with no arguments), should compile the program. An additional target, `clean`, should delete any files created when compiling the program (typically just the compiled program).

You may create this makefile using a text editor of your choice.

A typical makefile would be:

```
learn: learn.c
	gcc -Wall -Werror -fsanitize=address -o learn learn.c

clean:
	rm -f learn
```

Note that the command for compiling `learn` uses GCC warnings and the address sanitizer. You will lose one style point if your makefile does not include these.

Note that the makefile format requires tab characters on the indented lines. Text copied from this document may not be formatted correctly.

3.3 Creating the archive

We will use `tar` to create the archive file. To create the archive, first ensure that your `src` directory contains only the source code and makefile needed to compile your project. Any compiled programs, object files, or other additional files must be moved or removed.

Next, move to the directory containing `src` and execute this command:

```
tar czvf pa2.tar src
```

`tar` will create a file `pa2.tar` that contains all files in the directory `src`. This file can now be submitted through Sakai.

To verify that the archive contains the necessary files, you can print a list of the files contained in the archive with this command:

```
tar tf pa2.tar
```

3.4 Using the auto-grader

We have provided a tool for checking the correctness of your project. The auto-grader will compile your programs and execute them several times with different arguments, comparing the results against the expected results.

Setup The auto-grader is distributed as an archive file `pa2_grader.tar`. To unpack the archive, move the archive to a directory and use this command:

```
tar xf pa2_grader.tar
```

This will create a directory `pa2` containing the auto-grader itself, `grader.py`, a library `autograde.py`, and a directory of test cases `data`.

Do not modify any of the files provided by the auto-grader. Doing so may prevent the auto-grader from correctly assessing your program.

You may create your `src` directory inside `pa2`. If you prefer to create `src` outside the `pa2` directory, you will need to provide a path to `grader.py` when invoking the auto-grader.

Usage While in the same directory as `grader.py`, use this command:

```
python grader.py
```

The auto-grader will compile and execute the program `learn` in a directory `src` contained in the current working directory.

By default, the auto-grader will not print the output from your program, except for lines that are incorrect. To see all program output, use the `-v` option:

```
python grader.py -v
```

If you prefer not to see any output, use the `-q` option.

You may also use the auto-grader to check an archive before submitting. To do this, use the `-a` option with the archive file name. For example,

```
python grader.py -a pa1.tar
```

This will unpack the archive into a temporary directory, grade the programs, and then delete the temporary directory. This may be useful as a final check before (or just after) submitting your archive.

If your `src` directory is not located in the same directory as `grader.py`, you may specify it using the `-s` option. For example,

```
python grader.py -s ../path/to/src
```

Note: before testing your program, the auto-grader will execute `make clean` and `make` in the program directory. If either command fails for any reason, such as compiler errors or a missing or invalid makefile, you will receive no points for the program. Before submitting, make sure that you can execute `make clean` and `make` in the program directory on an iLab machine.

3.5 Providing your own test cases

It is recommended that you create your own training and input files to test your project. If you wish, you may use the auto-grader to run some of these tests for you.

To use these tests, create a directory named `test` inside your `src` directory. For each test case, create a training input file, a data input file, and an *output reference* file. The first two follow the format described in Section 2. The output reference file has the same format as the output of `learn`.

All three files should have names beginning with `test.` and a unique identifier for the test, such as a number. The training input file will end with `.train.txt`, the data input file will end with `.data.txt`, and the reference output file will end with `.ref.txt`.

If the auto-grader detects appropriately named files in the `src/test` directory, it will execute those tests and report how many were successful (meaning that `learn` produced output identical to the reference output file).

This diagram shows the layout of a project that includes an additional test case:

```
src
+- Makefile
+- learn.c
+- test
   +- test.1.train.txt
   +- test.1.data.txt
   +- test.1.ref.txt
```

4 Grading

This project is worth 100 points, of which 80 points will be determined by program behavior on test cases and 20 will be determined by examination of source code.

The auto-grader provided for students includes several test cases, but additional test cases will be used during grading. Make sure that your programs meet the specifications given, even if no test

case explicitly checks it. It is advisable to perform additional tests of your own devising, including but not limited to the use of

4.1 Style points

Your program will receive up to 20 style points. These will be awarded by graders as part of a manual inspection of your code. The graders will consider several aspects of your program, such as organization and use of comments.

Some advice for winning points:

- Make sure to use `-Wall -Werror -fsanitize=address` when compiling.
- Use indentation to make your code more readable.
- Document your code with comments. At a minimum, describe the purpose of any functions you create.
- Do not put all your code in a single function.

This program can be broken into two layers: an inner layer that performs matrix operations, and an outer layer that reads the input and generates output. Being aware of these layers may help you organize your code into functions.

4.2 Academic integrity

You must submit your own work. You should not copy or even see code for this project written by anyone else, nor should you look at code written for other classes. We will be using state of the art plagiarism detectors. Projects which the detectors deem similar will be reported to the Office of Student Conduct.

Do not post your code on-line or anywhere publically readable. If another student copies your code and submits it, both of you will be reported.

5 Advice

Start working early. Before you do any coding, make sure you can run the auto-grader, create a Tar archive, and submit that archive to Sakai. You don't want to discover on the last day that `scp` doesn't work on your personal machine.

Decide your data structures and algorithms first. Make sure you know how you will represent your matrices, and what steps you need to take to multiply them, transpose them, and invert them. Writing out pseudocode is not required, but it may be a good idea.

Make it easy to test your code. If you write your matrix operations in different functions, you can write specialized main functions to test them until you are confident they are correct.

Start working early. You will almost certainly encounter problems you did not anticipate while writing this project. It's much better if you find them in the first week.