CS211 Fall 2023
Programming Assignment II

David Menendez

Due: Friday, October 27, 11:59 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 a 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.

We can relate the matrices X, Y, and W with this equation:

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

Our goal will be to estimate the prices Y' for some houses with attributes X'. This is easily done if we know the weights W, but we do not. Instead, we can observe the attributes X for houses with known prices Y. We will use a strategy known as *one-shot learning* to deduce W, given X and Y.

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* by calculating

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

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 $x_{i,k} + a x_{j,k}$.

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.

Algorithm 1 is the implementation of Gauss-Jordan your program must implement. Note that the only row operations used are multiplying (or dividing) a row by a number and adding (or

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

subtracting) a multiple of one row to another. Given a matrix M, we will use M_i to refer to row i of M and $M_{i,j}$ to refer to the number in row i, column j. For this assignment, we will start counting rows and columns from 0.

Algorithm 1 Simplified Gauss-Jordan elimination

```
procedure INVERT(M : n × n matrix)
    N ← n × n identity matrix
    for p ← 0, 1, ··· , n − 1 do
        f ← M_{p,p}
        divide M_p by f
        divide N_p by f
        for i ← p + 1, ··· , n − 1 do
            f ← M_{i,p}
            subtract M_p × f from M_i
            subtract N_p × f from N_i
        end for
    end for
    for p ← n − 1, ··· , 0 do
        for i ← p − 1, ··· , 0 do
            f ← M_{i,p}
            subtract M_p × f from M_i
            subtract N_p × f from N_i
        end for
    end for
    return N
end procedure
```

To illustrate Gauss-Jordan, we will walk through the process of inverting the matrix

$$M = \begin{bmatrix} 1 & 2 & 4 \\ 1 & 6 & 8 \\ 1 & 1 & 6 \end{bmatrix}.$$

As a notational convenience, we will create an *augmented* matrix $A = M|I$ by adjoining the identity matrix I to M.

$$A = \left[\begin{array}{ccc|ccc} 1 & 2 & 4 & 1 & 0 & 0 \\ 1 & 6 & 8 & 0 & 1 & 0 \\ 1 & 1 & 6 & 0 & 0 & 1 \end{array}\right]$$

It is not necessary for your program to create A; instead, you can represent the two sides of A as two separate matrices.

The goal of Gauss-Jordan is to turn the left half of A into an identity matrix by applying row operations. At each step, we will identify a particular row as the *pivot row*. The element that lies on the diagonal (that is, element $A_{p,p}$) is the *pivot element*.

The first step is to turn the M 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

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

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

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

To zero out $A_{2,1}$, we add A_1 to A_2.

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

Now the pivot row is A_2. To make $A_{2,2} = 1$, we divide row A_2 by 3.

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 4 & 1 & 0 & 0 \\ 0 & 1 & 1 & -\frac{1}{4} & \frac{1}{4} & 0 \\ 0 & 0 & 1 & -\frac{5}{12} & \frac{1}{12} & \frac{1}{3} \end{array}\right].$$

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 A_2 from A_1, yielding

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 4 & 1 & 0 & 0 \\ 0 & 1 & 0 & \frac{1}{6} & \frac{1}{6} & -\frac{1}{3} \\ 0 & 0 & 1 & -\frac{5}{12} & \frac{1}{12} & \frac{1}{3} \end{array}\right].$$

Then we subtract $4 \times A_2$ from A_0, yielding

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 0 & \frac{8}{3} & -\frac{1}{3} & -\frac{4}{3} \\ 0 & 1 & 0 & \frac{1}{6} & \frac{1}{6} & -\frac{1}{3} \\ 0 & 0 & 1 & -\frac{5}{12} & \frac{1}{12} & \frac{1}{3} \end{array}\right].$$

Now the elements above $A_{2,2}$ are 0.

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

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & \frac{7}{3} & -\frac{2}{3} & -\frac{2}{3} \\ 0 & 1 & 0 & \frac{1}{6} & \frac{1}{6} & -\frac{1}{3} \\ 0 & 0 & 1 & -\frac{5}{12} & \frac{1}{12} & \frac{1}{3} \end{array}\right].$$

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 more detailed 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) $c \times A_i$ to (or from) A_j?

You MUST use the algorithm as described. 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 `estimate` 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. `estimate` 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 `estimate`:

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

Implementation notes The description of Gauss-Jordan elimination given in section 1.1 uses an augmented matrix with twice as many columns as the input matrix X. This is an illustrative tool, and not meant as an implementation requirement. It is simpler to use two matrices that begin as X and the identity matrix I and apply identical row operations to both, until the first matrix becomes I and the second is X^{-1}.

It is recommended to write separate functions to compute the inverse of a matrix, the transpose of a matrix, and the product of two matrices. You may find it simpler to avoid allocating memory within these functions; instead, pass them the input matrix or matrices and a pre-allocated matrix that will be used for the output.

Having separate functions will simplify your development, as you can test your implementations of each separately.

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

`estimate` 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, `estimate` MAY print "error" and return exit code 1. Your code will not be tested with these scenarios.

3 Grading

Your submission will be awarded up to 100 points, based on how many test cases your program completes successfully.

The auto-grader provided for students includes half of the test cases that will be used during grading. Thus, it will award up to 50 points.

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.

3.1 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.

4 Submission

Your solution to the assignment will be submitted through Canvas. You will submit a Tar archive file containing the source code and makefiles 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 makefiles, how to create the archive, how to use the provided auto-grader, and how to create your own test files to supplement the auto-grader.

4.1 Directory structure

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

This diagram shows the layout of a typical project:

```
src
+- Makefile
+- estimate.c
```

Note that your code and makefile go directly in `src`, without any subdirectories.

4.2 Makefiles

We will use `make` to manage compilation. Your `src` directory will contain a file named `Makefile` that describes at least two targets. The first target must compile the program. An additional target, `clean`, must delete any files created when compiling the program (typically just the compiled program).

The auto-grader script is distributed with an example makefile, which looks like this (note that an actual makefile must use tabs rather than spaces for indentation):

```
TARGET = estimate
CC     = gcc
CFLAGS = -g -std=c99 -Wall -Wvla -Werror -fsanitize=address,undefined

$(TARGET): $(TARGET).c
        $(CC) $(CFLAGS) $^ -o $@

clean:
        rm -rf $(TARGET) *.o *.a *.dylib *.dSYM
```

It is simplest to copy this file into your `src` directory, renaming it `Makefile`.

It is further recommended that you use `make` to compile your programs, rather than invoking the compiler directly. This will ensure that your personal testing is performed with the same compiler settings as the auto-grader. The makefiles created in the build directory by the auto-grader refer to the makefiles you create in the source directory and therefore pick up any changes made.

You may add additional compiler options as you see fit, but you are advised to leave the compiler warnings, sanitizers, and debugger information (`-g`). The makefile shown here specifies the C99 standard, in order to allow C++-style `//` comments; you may change that to C89, if you prefer.

Compiler options The sample makefile uses the following compiler options, listed in the `CFLAGS` make variable:

-g Include debugger information, used by GDB and AddressSanitizer.

-std=c99 Require conformance with the 1999 C Standard. (Disable GCC extensions.) You may change this to `-std=c89` or `-std=c90` at you discretion.

-Wall Display most common warning messages.

-Wvla Warn when using variable-length arrays.

-Werror Promote all warnings to errors.

-fsanitize=address,undefined Include run-time checks provided by AddressSanitizer and UBSan. This will add code that detects many memory errors and guards against undefined behavior. (Note that these checks discover problems with your code. Disabling them will not make your code correct, even if it seems to execute correctly.)

Target and dependency variables Note the use of `$@` (indicating the target name) and `$^` (indicating the dependencies). The auto-grader uses some advanced features of `make` to put the source files and object files in different directories. If you prefer to write your own Makefile instead of using the sample, you will need to use these variables in order for the auto-grader to successfully compile your project. Contact me with any questions about how to do this.

4.3 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 makefiles 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:

```
pa2$ tar -czvf pa2.tar src/Makefile src/estimate.c
```

`tar` will create a file `pa2.tar` that contains your makefile and source code. (If you are using multiple source files, or have re-named your source code, you will need to adjust this command accordingly.) This file can now be submitted through Canvas.

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

```
pa2$ tar -tf pa2.tar
```

You should also use the auto-grader to confirm that your archive is correctly structured.

```
pa2$ ./grader.py -a pa2.tar
```

4.4 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`, a makefile template `template.make`, 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 `pa1`. If you prefer to create `src` outside the `pa1` directory, you will need to provide a path to `grader.py` when invoking the auto-grader (see below).

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

```
pa2$ ./grader.py
```

The auto-grader will compile and execute the program in the directory `src`, assuming `src` has the structure described in section 4.1. The compiled program will be in a directory `build`, which is created by the auto-grader.

During development, you may prefer to use the `--stop` or `-1` option, which produces more program output but stops after the first failed test case.

```
pa2$ ./grader.py -1
```

To obtain usage information, use the `-h` option.

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

```
pa2$ ./grader.py -v
```

To see program output for all tests, use `-vv`. To see no program output, use `-q`.

Disabling leak detection When run on an iLab machine, AddressSanitizer will automatically provide leak detection, printing an error message if any memory allocated by your program has not been freed by the time the program concludes. The auto-grader will report this as a failed test case.

To disable leak detection for a specific test run, use the option `--lsan off`.

```
pa2$ ./grader.py --lsan off
```

Checking your archive We recommend that you use the auto-grader to check an archive before submitting. To do this, use the `--archive` or `-a` option with the archive file name. For example,

```
pa2$ ./grader.py -a pa2.tar
```

This will unpack the archive into a temporary directory, grade the programs, and then delete the temporary directory.

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

```
pa2$ ./grader.py -s ../path/to/src
```

Refreshing the build directory In the unlikely event that your build directory has become corrupt or otherwise unusable, you can simply delete it using `rm -r build`. Alternatively, the `--fresh` or `-f` option will delete and recreate the build directory before testing.