CS 211: Computer Architecture, Fall 2017
Programming Assignment 1: Simple Learning

This assignment is designed to provide you some experience writing programs with the C programming language and to get a glimpse of how machine learning works. There is significant hype and excitement around artificial intelligence (AI) and machine learning (ML). For this assignment, you will write a C program that implements a simple machine learning algorithm for predicting house prices based on historical data.

The price of the house (y) can depend on various attributes: number of bedrooms (x1), total size of the house (x2), number of baths (x3), and the year the house was built (x4). If these are the only variables that affect price, then the price of the house can be computed by the following equation:

$$y = w0 + w1 * x1 + w2 * x2 + w3 * x3 + w4 * y4 \quad (1)$$

Given a house, we might know the attributes of the house (i.e., x1 , x2 , x3 , x4), however, we don't know the weights for these attributes: w0 , w1 , w2 , w3 and w4. For instance, having 2 baths may be more important to some people than others, so the weight of that factor can change based on the value people attach to it.

The goal of our machine learning algorithm is to learn the weights for the attributes of the house from lots of training data. Training data is data that is known (or presumed) to be fully diagnostic and representative of what we want to test that is used to 'teach' the machine algorithm. The algorithm will then take what it has 'learned' from the training data and later apply it to testing data, data where some of the factors and variables are not known, and will estimate the most likely values for those factors and variables.

Presume we have N examples in a training data set that provide the values of all attributes for N houses and the price. Let's say there are K attributes. We can represent the attributes from all the examples in the training data as a N x (K + 1) matrix as follows, which we call X:

```
[
        1       x0,0    x0,1    x0,2    x0,3
        1       x1,0    x1,1    x1,3    x1,3
        1       x2,0    x2,1    x2,2    x2,3
        1       x3,0    x3,1    x3,2    x3,3
        ...
        1       xn,0     xn,1   xn,2    xn,3
]
```

where n is N − 1. We can represent the prices of the house from the examples in the training data as a N x1 matrix, which we call Y:

```
[
        y0
        y1
        ..
        yn
]
```

Similarly, we can represent the weights for each attribute as a (K + 1) x1 matrix, which we call W:

[
    w0
    w1
    ..
    wk
]

The goal of our machine learning algorithm is to learn this weight matrix from the training data. Now in the matrix notation, entire learning process can be represented by the following equation, where X, Y , and W are matrices as described above:

$$X * W = Y \quad (2)$$

In particular, each weight times the variable for each parameter of a house equals the price, for all houses and their corresponding prices.

Using the training data, we can learn the weights using the below equation:

$$W = (X^T * X)^{-1} * X^T * Y \quad (3)$$

X^T is the transpose of the matrix X. X^{-1} is the inverse of the matrix X, so $(X^T * X)^{-1}$ is the inverse of $(X^T * X)$.

Your main task in this part to implement a program to read the training data and learn the weights for each of the attributes, which you will use to predict housing prices in the test data set.

In order to do this, you will have to implement functions to multiply matrices, transpose matrices, and compute the inverses of matrices.

While you don't need to understand the theory behindthis type of machine learning in order to write the code, you can learn more about it at https://www.youtube.com/watch?v=FIbVs5GbBlQ&hd=1. In particular, the algorithm you are computing is known as 'linear regression', in particular linear regression with the least square error as the error measure (often abbreviated in speech as “Lin Least Squares”) and is used in many fields. The matrix $(X^T * X)^{-1} * X^T$ is known as the 'pseudo-inverse' of X and is useful in many computational methods.

Computing the Inverse using Gauss-Jordan Elimination

To compute the weights above, your program has to compute the inverse of matrix. There are numerous methods to compute the inverse of a matrix. For this project you should compute the inverse using Guass-Jordan elimination, described below. If you compute inverse using any other method, you will risk losing all points for this part.

An inverse of a square matrix A is another square matrix B, such that A*B = B*A = I, where I is the identity matrix. This is similar to the idea of arithmetic inverses; 2/1 * 1/2 = 1. The identity matrix I is a matrix such that, for any square matrix, A: A * I = A. An identity matrix is a matrix of 0s with 1s on the diagonal.

Gauss-Jordan Elimination for computing inverses
Below, we give a sketch of Gauss-Jordan elimination method. Given a matrix A whose inverse needs to be computed, you first create a new 'augmented' matrix, called Aaug, by concatenating the identity matrix to the end of A, as shown below:

Matrix A:
[
1 2 4
1 6 7
1 3 2
]
The augmented matrix (Aaug) of A is:
[
1 2 4 1 0 0
1 6 7 0 1 0
1 3 2 0 0 1
]

That is the easy part. The next part is to perform arithmetic operations on the rows of the augmented matrix to change the part that was A (the first 3 columns, in this case) in to an identity matrix. In other words, to make the first 3 columns look like the last 3 columns.

The row operations you are allowed to use to compute the inverse for this assignment are:
• You can divide the entire row by a constant
• You can subtract a row by another row
• You can subtract a row by another row multiplied by a constant

You are not allowed to swap the rows. Be careful! In the traditional Gauss-Jordan elimination method, you are. Do not swap rows in this assignment.

Below is an example of computing the inverse of A using Aaug and the row operations detailed above.

Since our goal is to transform A part of the augmented matrix into an identity matrix, let's start at the first element, Aaug[0][0] is 1, good! That is what it should be for an identity matrix. Next row. Aaug [1][0] is not 0, but 1. This needs to change. There is a 1 in the first column of the first row (we just made sure of that), so we will subtract the first row from the second row because we want to make Aaug [1][0] == 0. Hence, we perform the operation $R1 = R1 - R0$, where R1 and R0 represents the second and first row of the augmented matrix. Augmented matrix Aaug after $R1 = R1 - R0$
[
1 2 4 1 0 0
0 4 3 -1 1 0
1 3 2 0 0 1
]

Now we want to make Aaug [1][1] == 1. Hence, we perform the operation R1 = R1 /4. The augmented matrix Aaug after R1 = R1 /4 is:

[

1	2	4	1	0	0
0	1	3/4	-1/4	1/4	0
1	3	2	0	0	1

]

Next, we want to make Aaug [2][0] = 0. Hence, we perform the operation R2 = R2 − R0 . The augmented matrix Aaug after R2 = R2 − R0 is:

[

1	2	4	1	0	0
0	1	3/4	-1/4	1/4	0
0	1	-2	-1	0	1

]

Next, we want to make Aaug [2][1] == 0. Hence, we perform the operation R2 = R2 − R1 . The augmented matrix Aaug after R2 = R2 − R1 is:

[

1	2	4	1	0	0
0	1	3/4	-1/4	1/4	0
0	0	-11/4	-3/4	-1/4	1

]

Now, we want to make Aaug [2, 2] == 1, Hence, we perform the operation R2 = R2 * -4/11 . The augmented matrix Aaug after R2 = R2 * -4/11 is:

[

1	2	4	1	0	0
0	1	3/4	-1/4	1/4	0
0	0	1	3/11	1/11	-4/11

]

Next, we want to make Aaug [1, 2] == 0, Hence, we perform the operation R1 = R1 − 3/4 * R2. Aaug then becomes:

[

1	2	4	1	0	0
0	1	0	-5/11	2/11	3/11
0	0	1	3/11	1/11	-4/11

]

Next, we want to make Aaug [0, 2] == 0, Hence, we perform the operation R0 = R0 − 4 ∗R2 . Aaug then becomes:

[

1	2	0	1/11	-4/11	16/11
0	1	0	-5/11	2/11	3/11
0	0	1	3/11	1/11	-4/11

]

Next, we want to make Aaug [0, 1] = 0, Hence, we perform the operation R0 = R0 − 2 ∗R1 .Aaug then becomes:

[

1	0	0	1	-8/11	10/11
0	1	0	-5/11	2/11	3/11
0	0	1	3/11	1/11	-4/11

]

At this time, the A part of the augmented matrix is an identity matrix and the inverse of the original matrix A is now the augmented part, the part that we originally tacked on the the back as an identity matrix:

[

1	-8/11	10/11
-5/11	2/11	3/11
3/11	1/11	-4/11

]

To understand why this is, think of how arithmetic inverses work. For regular arithmetic '1' is the identity value, since N * 1 = N, for any number. If you have any number X and want to compute its inverse, you want a number, Y such that: X * Y = 1. So that any modification made by X is reversed by its inverse, or to say it another way, multiplying by X and then Y is the same as multiplying by identity. The operations above start with matrix A and then record all operations you need to do to turn it into the identity matrix. Those recorded operations are then the inverse of A. If you can multiply by A and then by A inverse and you get the same result as having multipled by the identity matrix.

Great. Why do you care? Well, if you have a matrix of weights that are applied to a vector of parameters, you will get a vector of prices. The training data will be that, except for the weights. For a given set of houses with certain parameters, they sold for a certain amount. If you compute the inverse of this multiplication, you can discover the weights that were applied to get from parameters to the prices - and then you can apply those weights to new data.

Input/Output specification

Usage interface

Your program learn will be executed as follows:

./learn <train-data-file-name> <test-data-file-name>

where <train-data-file-name> is the name of the training data file with the attributes and price of a house. You can assume that the training data file will exist and that it is well structured. The <test-data-file-name> is the name of the test data file with only the attributes of the house. You have to predict the price of the house for each entry in the test data file.

Input specification

The input to the program will be a training data file and a test data file.

Training data files

The first line in the training file will be an integer that provides the number of attributes (K) in the training set. The second line in the training data file will be an integer (N) providing the number of training examples in the training data set. The next N lines represent the N training examples. Each line for the example will be a list of comma-separated double precision floating point values. The first K double precision values represent the values for the attributes of the house. The last double precision value in the line represents the price of the house.

An example training data file (train1.txt) is shown below:

```
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
```

In the example above, there are 4 attributes and 7 training data examples. Each example has values for the attributes and last value is the price of the house. To illustrate, consider the training example below:

3.000000,1.000000,1180.000000,1955.000000,221900.000000

The first attribute has value 3.000000, the second attribute has value 1.000000, third attribute has value 1180.000000, and the fourth attribute has value 1955.000000. The price of the house for these set of attributes is provided as the last value in the line: 221900.000000

Testing data files

The first line in the testing file will be an integer (M) that provides the number of test data points in the file. Each line will have K attributes. The value of K is defined in the training data file. Your goal is predict the price of house for each line in the test data file. The next M lines represent the M test points for which you have to predict the price of the house. Each line will be a list of comma-separated double precision floating point values. There will be K double precision values that represent the values for the attributes of the house. An example test data file (test1.txt) is shown below:

```
2
3.000000,2.500000,3560.000000,1965.000000
2.000000,1.000000,1160.000000,1942.000000
```

It indicates that you have to predict the price of the house using your training data for 2 houses. The attributes of each house is listed in the subsequent lines.

Output specification

Your program should print the price of the house for each line in the test data file. Your program should not produce any additional output. If the price of the house is a fractional value, then your program should round it to the nearest integer, which you can accomplish with the following printf statement:

```
printf("%0.0f\n", value);
```

where value is a double holding the the price of the house.
Your program should predict the price of the entry in the test data file by substituting the attributes and the weights (learned from the training data set) in Equation (1).
A sample output of the execution when you execute your program as shown below:

```
./learn train1.txt test1.txt
```

... should be:

```
737861
203060
```

Hints and suggestions

• You are allowed to use functions from standard libraries but you cannot use third-party libraries downloaded from the Internet (or from anywhere else). If you are unsure whether you can use something, ask us.

• We will compile and test your program on the iLab machines so you should make sure that your program compiles and runs correctly on these machines. You must compile all C code using the gcc compiler with the -Wall -Werror -fsanitize=address flags.

• You should test your program with the autograder provided with the assignment.

Submission
Your submission should be a tar file named PA1.tar. To create this file, put everything that you are submitting into a directory named PA1. Then, cd.. into the directory containing PA1 (that is, PA1's parent directory) and run the following command:

tar cvf PA1.tar PA1

To check that you have correctly created the tar file, you should copy it into an empty directory and run the following command:

tar xvf PA1.tar

This should create a directory named PA1 with all your project files in it.

The PA1 directory in your tar file must have:
• Makefile: There should be at least two rules in your Makefile:
    learn: build your learn executable.
    clean: prepare for rebuilding from scratch.

• source code: all source code files necessary for building your programs. Your code should contain at least two files: learn.h and learn.c.

Grading guidelines
We will automatically build a binary using the Makefile and source code that you submit, and then test the binary for correct functionality and efficiency against a set of inputs.

• We should be able build your program by just running make.

• Your program should follow the format specified above for both both the parts.

• Your program should strictly follow the input and output specifications mentioned.
Note: This is perhaps the most important guideline: failing to follow it might result in you losing all or most of your points for this assignment. Make sure your program's output format is *exactly* as specified. Any deviation will cause the automated grader to mark your output as "incorrect". Requests for re-evaluation of programs rejected due to improper formatting will not be entertained.

• We will check all solutions pair-wise from all sections of this course (including those of parallel lectures) to detect cheating using computational lingusitc software and related tools. If two submissions are found to be similar, they will instantly be awarded zero points and reported to office of student conduct. See Rutgers CS's academic integrity policy at: https://www.cs.rutgers.edu/academic-integrity/introduction.

Autograder
We provide the AutoGrader to test your assignment. AutoGrader is provided as autograder.tar. Executing the following command will create the autograder folder:

tar xvf autograder.tar

There are two modes available for testing your assignment with the Autograder.
First mode
Testing when you are writing code with a PA1 folder:
1. Lets say you have a PA1 folder with the directory structure as described in the assignment.
2. Copy the folder to the directory of the autograder
3. Run the autograder with the following command

python PA1 autograder.py

It will run the test cases and print your scores.

Second mode
This mode is to test your final submission (i.e, PA1.tar)
1. Copy PA1.tar to the autograder directory
2. Run the autograder with PA1.tar as the argument. The command line is:

python PA1 autograder.py PA1.tar

The autograder will decompress your tar file and run the check code against it.