CS 211 HW 2 - Machine Learning with C

Fall 2020

1 Introduction

There is a significant hype and excitement around artificial intelligence (AI) and machine learning (ML). This assignment is designed to give you a glimpse of AI/ML. You'll write a C program to implement a simple "one-shot" machine learning algorithm to predict house prices based on historical data.

2 Problem Statement

There are several factors needs to be considered to determine the price of a house. For example, the price of the house (y) can depend on certain attributes of the house: number of bedrooms (x_1), total size of the house (x_2), number of baths (x_3), and the year the house was built (x_4). Then, the price of the house can be computed by the following equation:

$$y = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 \tag{1}$$

Given a house, we know the attributes of the house (i.e., x_1, x_2, x_3, x_4). However, we don't know the weights for these attributes: w_0, w_1, w_2, w_3 and w_4. The goal of the machine learning algorithm in our context is to learn the weights w_i from lots of training data.

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

$$\begin{pmatrix} 1 & x_{0,1} & x_{0,2} & ... & x_{0,\mathrm{k}} \\ 1 & x_{1,1} & x_{1,2} & ... & x_{1,\mathrm{k}} \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ 1 & x_{\mathrm{n\text{-}1,1}} & x_{\mathrm{n\text{-}1,2}} & ... & x_{\mathrm{n\text{-}1,k}} \end{pmatrix}$$

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 x 1 matrix Y , of the form

$$\begin{pmatrix} y_0 \\ y_1 \\ . \\ . \\ . \\ y_{\mathrm{n\text{-}1}} \end{pmatrix}$$

where each row gives the price of a house. Finally, the weights will be a (k + 1) x 1 matrix W, of the form

$$\begin{pmatrix} w_0 \\ w_1 \\ . \\ . \\ . \\ w_{\mathrm{k+1}} \end{pmatrix}$$

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, and calculate

$$W = (X^T X)^{-1} X^T Y$$

where X^T is the transpose of X and $X^T X$ 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 $X'W = Y'$.

3 Algorithm

You'll need to compute $(X^T X)^{-1} X^T Y$ in order to learn W for the given matrices X and Y. This process requires the following matrix operations:

- Multiplication
- Transpose (Transposing an m x n matrix produces an n x m matrix where each row of the X becomes a column of X^T)
- Inversion (To find the inverse of $X^T X$, you will use a simplified form of Gauss-Jordan elimination)

3.1 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 $x_{i,k} + ax_{j,k}$

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

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
```

Figure 1: Pseudocode for Gauss-Jordan Elimination: 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.

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

An Example

We begin with the matrix we wish to invert:

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

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 1/4).

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

Then, we subtract A_2 from A_3, yielding

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

Next, we divide A_2 by -11/4, yielding

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

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 $3/4A_2$ from A_1, yielding

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

Then we subtract $4A_2$ from A_0, yielding

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

Now the elements above $A_{2,2}$ are 0. Next, we subtract $2A_1$ from A_0, yielding

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

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.

Tips

- Write this algorithm in pseudocode before you begin implementing it in C
- Try using it to invert a small square matrix and understand the operations performing at each step

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 acceptable to create and manipulate an augmented matrix and to extract M^{-1} from it later.

Caveat
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.

4 Program Specification

You can run your program **ml** as follows:

./ml train-data-file-name test-data-file-name

where `train-data-file-name` is the name of the training data file with attributes and price of the 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 attributes of the house. You have to predict the price of the house for each entry in the test data file.

Structure of the training data file 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
- The 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 which is 221900.000000

Structure of the test data file the first line in the training 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("%0.0lf\n", value);
```

where value is the price of the house and its type is double in C. 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:

./ml 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
- 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 -fsanitize=address` flags and therefore you are not allowed to modify the Makefile you are provided with the autograder

5 Submission

You are provided a tar file named hw2-autograder.tar. Once you untar (**$ tar xvf hw2-autograder.tar**) it, you'll have a directory hw2-autograder as follows:

```
hw2-autograder
├── hw2
│   ├── Makefile
│   └── ml.c
├── testcases
└── autograder.py
```

All you need to modify the ml.c file under hw2 directory. Once you are done with your ml.c file, you can simply tar the hw2 directory (**$ tar cvf hw2.tar hw2**) and submit it.

5.1 Autograder

There are two ways to run the autograder to test your program.

First Mode is to test the program when you are writing code with a hw2 folder:

$ python autograder.py

It will run the test cases from the testcases directory and print your scores.

Second Mode is to test your final submission (i.e, hw2.tar)

$ python autograder.py hw2.tar

This command will test your hw2.tar file and print the scores. If you tar the same directory you tested in the First Mode then both scores should be the same.

References/Help
You may find these following materials useful:

- The Pseudo Inverse
  https://youtu.be/FIbVs5GbBlQ?list=PLD63A284B7615313At=2440
- Inverse of a Matrix
  https://www.mathsisfun.com/algebra/matrix-inverse.html