CSC 4780/6780
Fall 2022
Homework 03

September 8, 2022

This homework is due at 11:59 pm on Sunday, Sept 11. It must be uploaded to iCollege by then. No credit will be given for late submissions. A solution will be released by noon on Monday, Sept 12.

Once again: it is a good idea to get this done and turn it in early.

1 What are we doing?

Linear regression is a very common method of making predictions. You should learn both ways of solving for the coefficients: matrix inversion and gradient descent.

You have decided to create a company called ”Zillom” that estimates the price that a house will sell for. I have given you a spreadsheet (`properties.xlsx`) with the features and prices of 519 houses that have sold recently in Cleveland. The first five columns are the features you will use to predict prices:

- `sqft_hvac`: Indoor square footage
- `sqft_yard`: Outdoor square footage
- `bedrooms`: Number of bedrooms in the house
- `bathrooms`: Number of bathrooms in the house
- `miles_to_school`: Number of children would need to walk to the nearest elementary school

You are going to use this spreadsheet (and linear regression!) to create a formula for predicting the sale price of any house in Cleveland.

(I got Stable Diffusion running, and I asked it to make ”an Edward Hopper painting of a realtor in front of a modern house” for you. I've included three of the images in this document.)

2 Write programs that do linear regression

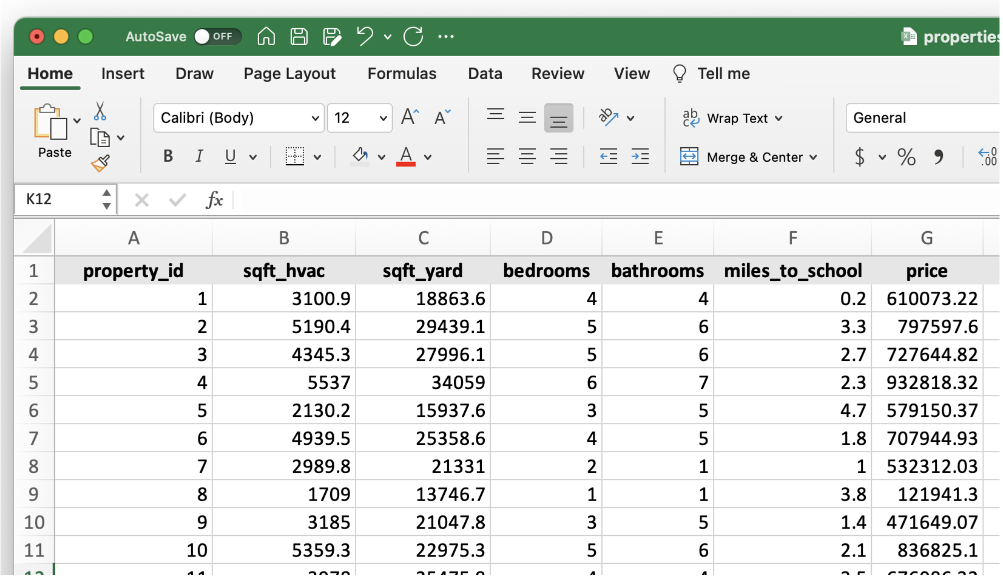
You are going to create three python programs:

- `linreg_mi.py` uses matrix inversion to come up with the formula.
- `linreg_sckit.py` uses scikit-learn to find the formula.
- `linreg_gd.py` uses gradient to converge upon the formula.

All three take the filename of the spreadsheet as an argument:

```
> python3 linreg_mi.py properties.xlsx
```

The program will read the excel spreadsheet that has the features of houses and the price they sold for:



	A	B	C	D	E	F	G
1	property_id	sqft_hvac	sqft_yard	bedrooms	bathrooms	miles_to_school	price
2	1	3100.9	18863.6	4	4	0.2	610073.22
3	2	5190.4	29439.1	5	6	3.3	797597.6
4	3	4345.3	27996.1	5	6	2.7	727644.82
5	4	5537	34059	6	7	2.3	932818.32
6	5	2130.2	15937.6	3	5	4.7	579150.37
7	6	4939.5	25358.6	4	5	1.8	707944.93
8	7	2989.8	21331	2	1	1	532312.03
9	8	1709	13746.7	1	1	3.8	121941.3
10	9	3185	21047.8	3	5	1.4	471649.07
11	10	5359.3	22975.3	5	6	2.1	836825.1

(property_id will be the index for the dataframe; you ignore it in the calculations.)

All three will find the hyperplane that minimizes the L2 error for those 519 data points. Each program will output those coefficients as a formula for predicting house prices:

```
predicted price = $32,362.85 + ($85.61 x sqft_hvac) + ($2.73 x sqft_yard) +
                                   ($59,195.07 x bedrooms) + ($9,599.24 x bathrooms) +
                                   ($-17,421.84 x miles_to_school)}
```

They will also output the R^2 score for fit. What is R^2?

3 R^2

We usually speak of the inputs for a prediction as the matrix X where each row x_i is the input for one data point.

We usually speak of the vector of correct answers (”the ground truth”) as Y where each element y_i is the output for one data point. The mean of Y is usually denoted $\bar{y}$.

Your linear regression will create a set of coefficients B. For each input x_i, you can use B to create a prediction $\hat{y}_i$.

The dumbest linear function for estimating would be just the constant function that returned the mean of Y. This would be equivalent to asking ”How much will this house sell for?” and getting the answer, ”Well, I’m going to ignore all the features of the house, and tell you that the average price of these 519 houses is $603,139.95.” The sum of squared errors for this dumb approach would be

$$\sum_{i=1}^{n} (y_i - \bar{y})^2$$

Your predictions $\hat{y_i}$ should have a smaller sum of squared errors:

$$\sum_{i=1}^{n} (y_i - \hat{y_i})^2$$

The R^2 score for a set of predictions is:

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y_i})^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$

If the data is basically linear without much noise, R^2 will be close to 1: you have good fit.

If the data is not linear or very noisy, R^2 will be close to 0: your fit is terrible, about as bad as ignoring all the features and just using the mean.

4 Steps

You will edit two files `util.py` and `linreg_gd.py`.

4.1 util.py

You need to write three functions in `util.py`. When they are done correctly, `linreg_mi.py` and textttlinreg_scikit.pywill run unchanged. The three functions are:

- `read_excel_data` which reads in the excel file and returns `X`, `Y`, and `labels`. `Y` is a 1-dimentional numpy array containing the last column of the spreadsheet. `X` is a 2-dimensional numpy array that contains the data in the other columns *and* the first column is filled with 1s. `labels` is a list of strings from the header in the spreadsheet.
- `format_prediction` which takes `B` (the vector of coefficients) and `labels` that you created in `read_excel_data`. Then it returns a string like this:

  ```
  predicted price = $32,362.85 + ($85.61 x sqft_hvac) + ($2.73 x sqft_yard) +
                                       ($59,195.07 x bedrooms) + ($9,599.24 x bathrooms) +
                                       ($-17,421.84 x miles_to_school)}
  ```
- `score` that takes `B`, `X`, and `Y` and returns the R^2 score.

When `util.py` is done, you should be able to run `linreg_mi.py` and `linreg_scikit.py`.

4.2 linreg_gd.py

In this, you will be using gradient descent to minimize the squared error. The result should be very nearly the same as `linreg_mi.py`.

The features in `properties.xlsx` are on very different scales (2 bathrooms vs 50,000 square foot yards). As a result, converging would take a very, very long time if you don't first standardize the features.

The first step is to find the mean and the standard deviation of each column of `X`. Use those to make each feature have a mean of 0 and a standard deviation of 1.

Then start with a guess of zero for all the coefficients. Do the following many times:

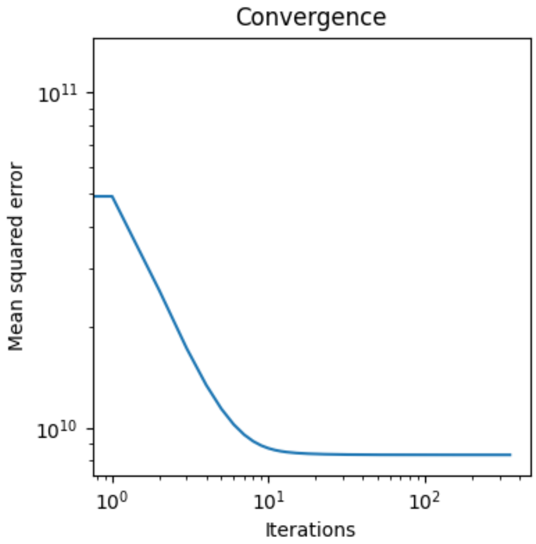
- Calculate the gradient
- Update your guess. (Multiply the gradient by -0.001 and add to the last guess.)
- Compute and record the new mean squared error

When the gradient gets small (and thus the changes to the coefficients gets small), stop. It should take a few hundred iterations.

The coefficients that you have calculated are for standardized inputs. Using the means and standard deviations you computed early, adjust them to use unstandardized data. (The math for this is in the next section.)

Calculate and display the R^2 score.

Plot the mean square error vs. iterations. This will be most interesting if you use log scaling for both the x and y axes. Save it as `err.png`. Mine looks like this:



4.3 Standardizing and compensating for standardizing

Using the matrix X, you will calculate the vector of means $M = [m_1, m_2, \ldots, m_d]$ and standard deviations $S = [s_1, s_2, \ldots, s_d]$.

Then you will create a new matrix X' that has normalized each entry. The entries in column j are given by

$$x'_j = (x_j - m_j)/s_j = \frac{x_j}{s_j} - \frac{m_j}{s_j}$$

Now the matrix X' has two nice properties:

- The mean of every column is 0.
- The standard deviation of every column is 1.

When you use those numbers to do linear regression, you will get a vector $B' = [b'_0, b'_1, b'_2, \ldots, b'_d]$ which can be used for predictions like this:

$$\hat{y} = b'_0 + b'_1 x'_1 + \ldots + b'_d x'_d$$

where the inputs have been standardized using the M and S that you calculated from the training data.

However, we really want the vector $B = [b_0, b_1, b_2, \ldots, b_d]$ so that we can put non-standardized data $[x_1, \ldots, x_d]$ into the formula

$$\hat{y} = b_0 + b_1 x_1 + \ldots + b_d x_d$$

Using the definition of x'_j from above we have:

$$\hat{y} = b'_0 + b'_1 \left(\frac{x_1}{s_1} - \frac{m_1}{s_1} \right) + \ldots + b'_d \left(\frac{x_j}{s_j} - \frac{m_j}{s_j} \right)$$

Expanding and sorting we get:

$$\hat{y} = \left(b'_0 - b'_1 \frac{m_1}{s_1} - \ldots - b'_d \frac{m_j}{s_j} \right) + \frac{b'_1}{s_1} x_1 + \ldots + \frac{b'_d}{s_j} x_j$$

Thus,

$$b_0 = b'_0 - b'_1 \frac{m_1}{s_1} - \ldots - b'_d \frac{m_j}{s_j}$$

and for $0 < j \leq d$:

$$b_j = \frac{b'_j}{s_j}$$

Use those for your final answer and the R^2 calculation.

5 Scikit-Learn

In a job situation, you will use the sklearn implementation 99% of the time. It uses Single Value Decomposition and psuedo-inverses, so it is usually faster and more reliable than the matrix inversion approach.

6 Criteria for success

If your name is Fred Jones, you will turn in a zip file called `HW03_Jones_Fred.zip` of a directory called `HW03_Jones_Fred`. It will contain:

- `linreg_mi.py` (You don't need to edit.)
- `linreg_scikit.py` (You don't need to edit.)

- linreg_gd.py (Add about 22 lines of code.)
- util.py (Add about 20 lines of code.)
- err.png
- properties.xlsx (You don't need to edit.)

Be sure to format your python code with black before you submit it.

We will run your code like this:

```
cd HW03_Jones_Fred
python3 linreg_mi.py properties.xlsx
python3 linreg_scikit.py properties.xlsx
python3 linreg_gd.py properties.xlsx
```

We expect the following output:

```
> python3 linreg_mi.py properties.xlsx
Read 519 rows, 5 features from 'properties.xlsx'.
predicted price = $32,362.85 + ($85.61 x sqft_hvac) + ($2.73 x sqft_yard) +
      ($59,195.07 x bedrooms) + ($9,599.24 x bathrooms) + ($-17,421.84 x miles_to_school)
R2 = 0.875699

> python3 linreg_scikit.py properties.xlsx
Read 519 rows, 5 features from 'properties.xlsx'.
predicted price = $32,362.85 + ($85.61 x sqft_hvac) + ($2.73 x sqft_yard) +
    ($59,195.07 x bedrooms) + ($9,599.24 x bathrooms) + ($-17,421.84 x miles_to_school)
R2 = 0.875699

> python3 linreg_gd.py properties.xlsx
Read 519 rows, 5 features from 'properties.xlsx'.
Took 352 iterations to converge
predicted price = $32,362.82 + ($85.61 x sqft_hvac) + ($2.73 x sqft_yard) +
     ($59,196.55 x bedrooms) + ($9,598.99 x bathrooms) + ($-17,421.85 x miles_to_school)
R2 = 0.875699
```

You will get 4 points for a well-written util.py that enables linreg_mi.py and linreg_scikit.py to get the right answer.

You will get 5 points for a well-written linreg_gd.py that uses gradient descent and gets approximately the same answer as linreg_mi.py.

You will get 1 points for a err.png that looks right.

Do this work by yourself. Stackoverflow is OK. A hint from another student is OK. Looking at another student's code is *not* OK.

7 Extra help

A good video on gradient descent and linear regression: `https://youtu.be/sDv4f4s2SB8`