DataMining Project

Thomas Broekman s1061435

December 2021

Abstract

The code for the project can be found on the following web address: https://github.com/ThomasB013/DMProject. In this project we see how to compute a best fit line with a method known as ordinary least squares (OLS). We will take a brief look at the math behind OLS and discuss the implementation afterwards. This document comes to its conclusion with a discussion of finding outliers using the OLS implementation.

Index

1. The goals of OLS
    1.1 Measuring errors
    1.2 Solution for OLS
    1.3 Confidence intervals

2. Implementation
    2.1 Matrix library and helper functions
    2.2 OLS implementation
    2.3 data_frame and user friendliness

3. Detecting outliers
    3.1 The algorithm
    3.2 The implementation

4. Examples
    4.1 Finding a best fit line.
    4.2 Successfully detecting outliers in a 2D data set.
    4.3 Wrongly detecting outliers in a 2D data set.

5. Sources

1 The goal of OLS

OLS, ordinary least squares is a method that tries to find a best fit line for the data in the training set. This allows you to draw conclusions on the influence of variables on each other. Suppose we have a data set containing the columns age, height, weight, bcpw (beers consumed per week). We might wonder, what is the influence of the columns on the column weight? OLS gives us a method do do this. In this example weight is the explained variable and age, height and bcpw are explanatory variables. Once the analysis is done we end up with an equation such as $weight = 0.01345 * bcpw - 0.03 * age + 0.32194 * height + 30$. Here 30 is called the intercept and the equation can be interpreted as "keeping other values constant, one centimeter of height is expected to add 321 grams to a person's weight."

1.1 Measuring errors

OLS uses the following variables:

- y, the actual value for y (explained variable)
- X, the observed data (explanatory variables)
- β, the coefficients for the observed data
- $u = y - X\beta$, the residual

Where y is a $n \times 1$ matrix consisting of n observed values. X is a $n \times k$ matrix consisting of n observations for k explanatory variables. β is a $k \times 1$ matrix, where $\beta_{i,1}$ corresponds to the coefficient of the ith explanatory variable, $1 \leq i \leq k$. Furthermore we will use the hat to indicate that something is estimated, for example $\hat{\beta}$ is an estimate of β, and the bar to indicate an average, for example $\hat{y}$ is the mean of y. The goal of OLS is to estimate $\hat{\beta}$ such that our prediction $\hat{y} = X\hat{\beta}$ is close to the actual value y. The following measures are of interest:

- $SST = \sum_{i=1}^{n}(y_i - \bar{y})^2$, total sum squares
- $SSE = \sum_{i=1}^{n}(\hat{y}_i - \bar{y})^2$, explained sum squares
- $SSR = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} \hat{u}^2$, residual sum squares

Note that we generally want $SSR < SST$, as this indicates that our predicted values are closer to the actual values than the mean. If this is not the case then we are better of always predicting the mean and throw away our prediction.

1.2 Solution for OLS

The key to understanding the solution is that our explanatory data spans a subspace in $\mathbb{R}^k$ and that our prediction will always be in this subspace, because we

assign coefficients to the columns of the explanatory data. Hence our predicted value is an orthogonal projection of y onto X.

$$\begin{aligned} y &= X\hat{\beta} + \hat{u} \\ X^T y &= X^T X\hat{\beta} + X^T\hat{u} \\ X^T y &= X^T X\hat{\beta} + 0 \\ \hat{\beta} &= (X^T X)^{-1} X^T y \end{aligned}$$

This formula imposes the restriction that we strictly work with numerical data and even worse we cannot recover from missing data. If a single observation is missing from an observation vector x_i, then the whole vector cannot be taken into account when estimating β unless we work with some default value generation. In return we get a deterministic algorithm and unbiased estimators, which we discuss in the next section.

1.3 Confidence intervals

In the previous section we concluded that we need the inverse of $X^T X$, this leads to the assumption that X has no perfect co-linearity. These are all the assumptions:

- MLR1. The model is $y = X\beta$.
- MLR2. The data is a random sample from the population.
- MLR3. No perfect co-linearity between explanatory columns.
- MLR4. $E(u_i|x_i) = 0$
- MLR5. $Var(u_i|x_i) = \sigma^2$
- MLR6. $u_i \sim N(0, \sigma^2)$

This leads to the following conclusions, for which the proofs are left as an exercise to the reader. But in all seriousness, proving this is out of the scope and too difficult for me. I will provide some intuition. From the assumptions it follows that

$$\begin{aligned} E(\hat{\beta}|X) &= E((X^T X)^{-1} X^T y|X) \\ &= E((X^T X)^{-1} X^T (X\beta + u)|X) \\ &= E((X^T X)^{-1} X^T X\beta + (X^T X)^{-1} X^T u)|X) \\ &= \beta + (X^T X)^{-1} X^T E(u|X) \\ &= \beta + 0 = \beta \end{aligned}$$

Therefore the estimator is unbiased. Moreover (without proof), $V(\hat{\beta}|X) = \sigma^2 (X^T X)^{-1}$. Note that when we have more data points $X^T X$ has bigger entries

and therefore the variance matrix is less, as it grows with the inverse of $X^T X$. Let d be a $1 \times k$ observation matrix, then

$$V(d\hat{\beta})) = \sigma^2 d(X^T X)^{-1} d^T \tag{1}$$

An unbiased estimator for the variance is

$$\hat{\sigma} = \sqrt{\frac{1}{n-k} \sum_{i=1}^{n} \hat{u}_i^2} \tag{2}$$

1.3.1 Testing for coefficients

An interesting question is if a certain data has a significant effect, this corresponds to "Is the coefficient nonzero for explanatory variable i?". To answer this we use

$$\frac{\hat{\beta}_j - \beta_j}{sd(\hat{\beta}_j)} \sim N(0,1) \tag{3}$$

But we don't know $sd(\hat{\beta}_j)$ as we can only estimate the standard deviation. Hence we need to use the estimated standard deviation for the jth coefficient, which is $\hat{\sigma}\sqrt{(X^T X)^{-1}_{j,j}}$. Where we use the square root of the jth diagonal entry of the estimated variance matrix for $\hat{\beta}$. To account for the added uncertainty we need to change our distribution to

$$\frac{\hat{\beta}_j - \beta_j}{esd(\hat{\beta}_j)} \sim t_{n-k} \tag{4}$$

Where t_{n-k} stands for the student t distribution with (n-k) degrees of freedom and esd for the estimated standard deviation. We can now compute the t-statistic T for our hypothesis $\beta_j = 0$. If that is unlikely, we reject the hypothesis that $\beta_j = 0$. Meaning that the jth column has a significant influence on the explained variable. We use a significant level of 0.05. Therefore we need that $|T| > t_{n-k;0.975}$. Note that this is a two sided test and the t-distribution is symmetric. Furthermore it follows that the 95% confidence interval for the coefficients is

$$\hat{\beta}_j - t_{n-k;0.975} esd(\hat{\beta}_j) \leq \beta_j \leq \hat{\beta}_j + t_{n-k;0.975} esd(\hat{\beta}_j) \tag{5}$$

The student-t distribution approaches $N(0,1)$ as the degrees of freedom parameter grows. So the distribution in equation 4 approaches a normal distribution. Note the similarity with the Central Limit Theorem.

1.3.2 Confidence intervals for predictions

In this subsection we answer the following: given our prediction $\hat{y}$, what is the interval such that we can say that the real y is in that interval with 95% confidence?

$$y_i - \hat{y}_i = x_i\beta + u_i - x_i\hat{\beta} = x_i(\beta - \hat{\beta}) + u_i \tag{6}$$

From MLR6 it follows $u_i \sim N(0, \sigma^2)$ and from equation 4 we have $\hat{\beta}_j - \beta_j \sim N(0, sd(\hat{\beta}_j))$. Together with the property of normal distributions that $N(\mu_1, \sigma_1^2) + N(\mu_2, \sigma_2^2) = N(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$ and equation 1 we obtain

$$x_i(\beta - \hat{\beta}) + u_i \sim N(0, \sigma^2 x_i (X^T X)^{-1} x_i^T + \sigma^2) \tag{7}$$

Here we also need to account for the estimation of the standard deviation. Therefore our 95% confidence interval is

$$\hat{y}_i - t_{n-k;0.975}\hat{\sigma}\sqrt{x_i(X^TX)^{-1}x_i^T + 1} \leq y_i \leq \hat{y}_i + t_{n-k;0.975}\hat{\sigma}\sqrt{x_i(X^TX)^{-1}x_i^T + 1} \tag{8}$$

2 Implementation

See https://github.com/ThomasB013/DMProject for the code of the implementation. I divided the code in interfaces, better known as header files (.h), and implementation (.cpp) files. Please scan through the header files in the Linear_regression folder before continuing.

2.1 Matrix library and helper functions

In the matrix directory I have coded a matrix for a general allocator and type. An allocator allocates memory on the heap. This is nice as it adds the possibility of reallocating during run time (for example by adding new columns). The matrix library consists of the following files:

- chm_iterator.h this defines a read-write and a read-only iterator, allowing several standard library functions to traverse the elements. CHM stands for consecutive homogenous memory. With that I mean all memory that consists of a single type for which the next element is simply at the location equal to current location plus the size of a single instance.
- mem_base.h this defines the idea of memory allocation. We can only move these memory chunks around. Copying is prohibited.
- my_vec.h is my own implementation of a vector.
- matrix.h is using the previous three directories. Note the using declaration for vector in the Matrix struct. I will refer to this by typing matrix::vector throughout the project. If I find that at some point my own vector is bugged I will simply replace this with std::vector at one place and this then allows me to still finish the project.

matrix is a rather unsafe type because anyone is allowed to add an element to a certain row at any given moment. To account for this I have defined errors at the top of the header file which can be thrown. For example:

```
matrix Data;
//Read some data.
matrix X;
//Read some X.
//Not sure if X is rectangular:
X.assert_rect("Something went wrong while reading X");
cout << Data*X; //Can safely assume rectangular matrix X.
```

The helper.h and helper.cpp files in the Linear_regression folder specialize on calculations with doubles, which represent decimal numbers in C++. Together they overload operators and defines functions to make the implementation of regression readable and easier to follow.

2.2 OLS implementation

For the implementation I made a class linear_regresser. While designing the interface I tried to somewhat follow the structure of the lab tutorials with python. Meaning that we make a new regresser class, and fit the data and then use the results. This class has public function such which allow the user to fit a data set. It has private attributes such as $(X^T X)^{-1}$ and β, I tried to keep a balance between not having to recalculate everything every time and not splitting the calculations up in too many different parts.

Another big goal while writing the code was to have a one to one relationship between the code in the source files and the formulas written in chapter 1 (or more generally speaking: the thought and intent behind the code should be directly reflected by the code). Furthermore we want to write readable code that is easy to interpret, update and change. As an example, Parse::Expression* sum() in parser.cpp does adhere to this. The constructor Token_stream::Token_Stream in the same file is of lesser quality in this context.

2.3 data_frame and user friendliness

An implementation of a data analysis algorithm also needs to be user friendly. For that I made a data-frame. This consists of a data matrix with column names. The main advantage of this is that a user no longer has to do trickery himself. Instead of referring to column 0 or 2 the user can simply refer to column "age" or "weight". Therefore data_frame will allow for commands such as

```
data_frame f{"data.txt"};
f.add_col("BMI", "weight/(height/100)^2");
f.regress("BMI", "age cal_p_day exc_p_week", std::cout);
```

The implementation of converting a string to a expression that can be evaluated row by row can be found in parser.cpp.

3 Detecting outliers

In this chapter we discuss an algorithm for detecting outliers. We use the notion that some observations can be odd in the sense that they are not what we expected given what we observed. Compare this to finding a point that is in a vacant space between the clusters by weighted clustering. If the point does not have any weights above a certain threshold it doesn't belong to any cluster, making it an outlier.

Let us first view an example to get the intuition right. A lot of people would find an adult that has a length of 1.20m odd. But when they see a child of 1.20m their minds don't think anything of it, why is that? Because given a person's age, everyone has their own range of 'normal' height. To put it more mathematically, $1.20 \notin NHI(25)$ but $1.20 \in NHI(10)$, where NHI is a function that returns a Normal Height Interval based on an input parameter age. For OLS, equation 8 defines our notion of odd. If an observed value is not in this interval it will be classified as an outlier.

3.1 The algorithm

Algorithm 1 Finding outliers with OLS

```
function FindOutliers(y, X, θ, maxIter)
    O ← ∅
    while (y, X) ≠ ∅ do
        R ← regress(y, X)
        for observation o in (y, X) do
            if o.y ∉ 95% confidence interval for o.x then
                O ← O ∪ o
                (y, X) ← (y, X) \ {o}
            end if
        end for
        if |removed observations| < θ or #iterations > maxIter then
            return O
        end if
    end while
    return O
end function
```

3.2 The implementation

The implementation follows the algorithm. It first adds a new vector of indexes to keep track of our original indices. It then uses the row_swap() and pop_back() to remove outliers in O(1), the same operations will be done on the vector of original ids. Note that we cannot add an ID column to X as we will regress on X. For bookkeeping we allocate a vector of integers with it's size equal to

the size of the observation data. The integers are initialized to zero and get set to the number of current iterations if they are outliers. This means that if an index is 0 at the end this does not represent an outlier and if it is positive it does represent an outlier. In the end this vector of integers is returned.

4 Examples

4.1 Finding a best fit line

https://www.princeton.edu/~otorres/Regression101.pdf refers to the data set used in this section. We only consider six columns, excluding the id column. The data from these columns can be found in Data/csat_data.txt. The output in these slides is slightly different because they are working with a keyword robust. Which indicates that they account for heteroskadacity. In that case MLR5 does not hold, meaning that variance of the error terms is no longer assumed constant. A real world example of this are the prices of stocks, their variance may increase after discovering a new covid variant but after we have seen that the variant is not damaging their variance decreases as the stocks stabilize. The commands can be found in main.cpp in the function csat_example(). I will shortly discuss the output of the regression (refer to the slides for more details):

```
Regression Ouput:
csat            expense         percent         income          high            college         cons
Coeff           3.352826e-03    -2.618177e+00   1.055852e-01    1.630841e+00    2.030894e+00    8.515649e+02
Std Dev         4.470883e-03    2.538491e-01    1.166094e+00    9.922470e-01    1.660118e+00    5.929228e+01
t-stat          7.499248e-01    -1.031391e+01   9.054611e-02    1.643584e+00    1.223343e+00    1.436215e+01
P > |t|         4.572030e-01    1.935119e-13    9.282551e-01    1.072320e-01    2.275677e-01    4.440892e-16
[95% Conf       -5.659356e-03   -3.129873e+00   -2.244968e+00   -3.692809e-01   -1.315489e+00   7.320465e+02
interval]       1.236501e-02    -2.106480e+00   2.456138e+00    3.630963e+00    5.377278e+00    9.710833e+02

# Obs: 51               # Reg: 6
SSE 1.846633e+05        SSR 3.935120e+04
SST 2.240145e+05
```

We tried to fit a line between *csat* and the other columns. The best-fit line we got can be read off the coefficients. The line is roughly

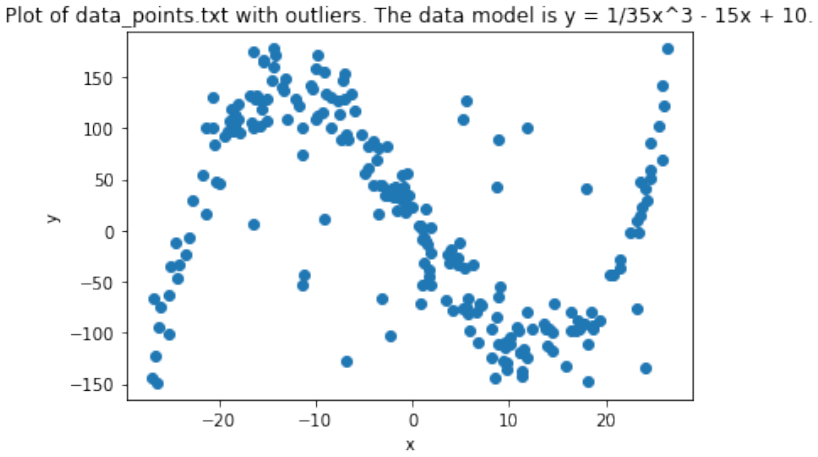
$$csat = 0.00353 \cdot expense - 2.62 \cdot percent + \\ 0.106 \cdot income + 1.63 \cdot high + 2.03 \cdot college + 852$$

We can also take a look at the row $P > |T|$. We see that the explanatory variables expense, income, high and college are not significant for a confidence level of 95%. We see that $SSR < SST$ by a large margin and hence our line is doing well (see 1.1). Although the performance on the training data is important, we would benefit from a second data set. We could use that data set for testing in order to get a better idea about the general performance of the estimator. Alternatively, we could have split up the data set into a training and testing part.

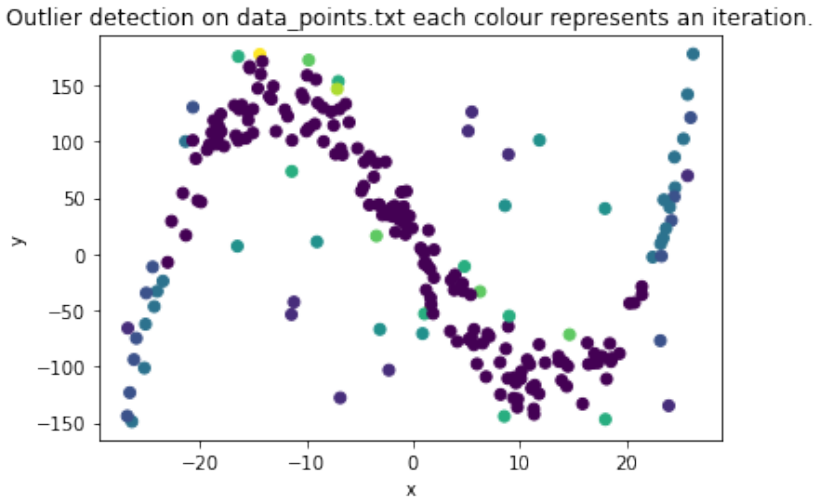
4.2 Successfully detecting outliers in a 2D data set

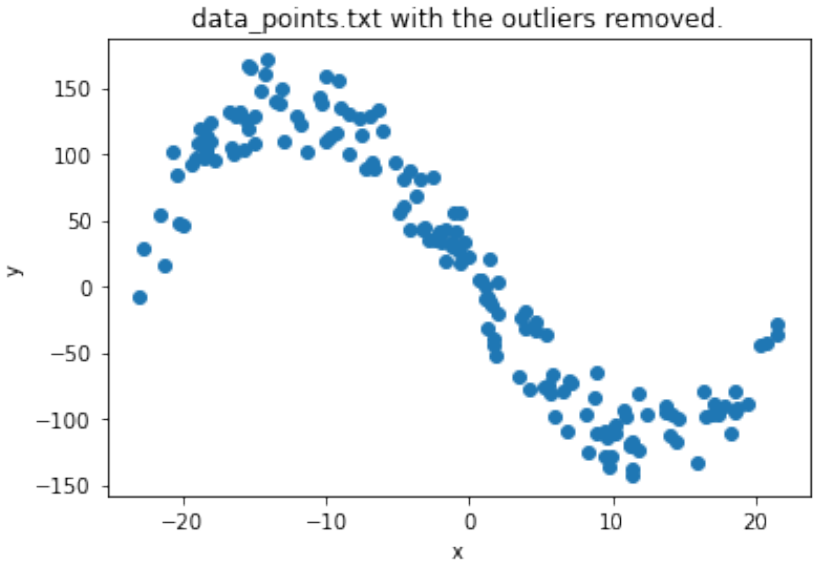
In this section we are going to detect outliers in a 2D data set. I generated the points around a line of $y = \frac{1}{35}x^3 - 15x + 10$, see gen_data_points in main.cpp.

The dataset generated is as follows:



We now try to get the outliers when using the correct model, a third degree polynomial. For the code of this example see Data/Outlier.ipynb and in main.cpp the function outlier_correct_example. The output we get is visualized in python notebook:

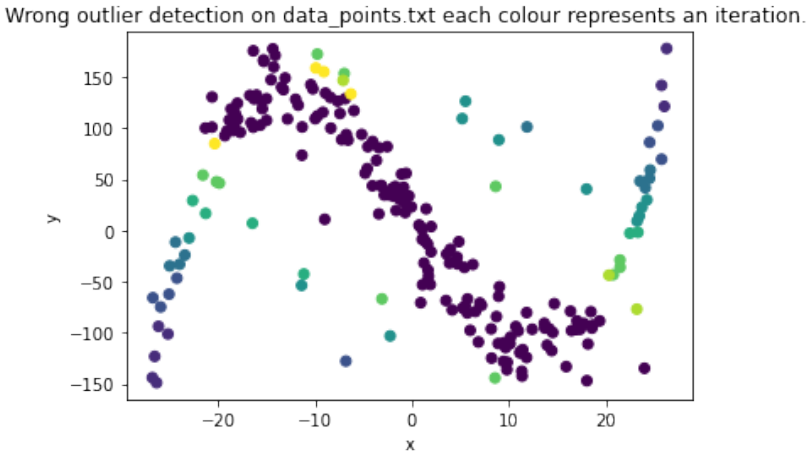


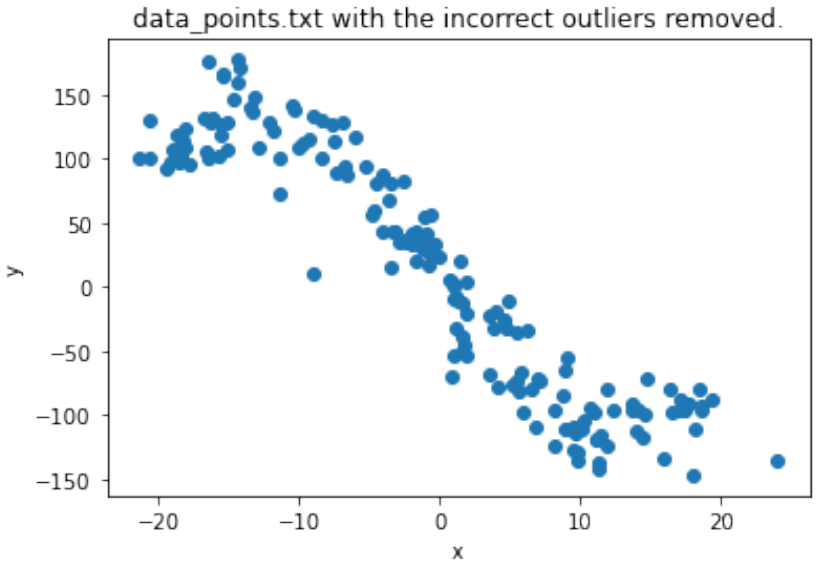


The algorithm successfully removed outliers. The core is very well represented, but we did lose some information in the tails. The performance might be improved by selecting a better θ or $maxIter$ (see Algorithm 1 in section 3.1)

4.3 Wrongly detecting outliers in a 2D data set

This section illustrates a mistake that seems obvious in 2D, but can be harder to spot in more dimensions. Here we try to fit a model that is not the real model behind the data. In this case we try to fit a linear model $y = ax+b$ Note that this also goes against MLR1. For the code see the function outlier_incorrect_example.





In this case we completely lose the tails. If we were to do the rest of our analysis only on this data set, supposedly with the outliers removed, we would likely get very bad results. Be very aware that if using Algorithm 1, you get a tunnel vision on your model and there is no way back to the real distribution behind the data.

5 Sources

The information for this document I got in the following way:

- For chapter 1 I studied several slides sent to me by a friend who is studying Econometrics: (https://www.tilburguniversity.edu/education/bachelors-programs/econometrics-and-operations-research/program-and-courses). First Year Content → Unit 4 → Introduction Econometrics.
- For chapter 2 I used https://cplusplus.com/reference/. The mem_base.h header is copied from The C++ Programming Language by Bjarne Stroustrup, section 13.6.2 "Representing Memory Explicitly". My_vec from my_vec.h is partially copied from subjects discussed in this book. For the function student_t_0_975 I used https://www.sjsu.edu/faculty/gerstman/StatPrimer/t-table.pdf
- The idea discussed in chapter 3 came to mind while reading about various data mining algorithms from the course book Introduction to Data Mining Second Edition.
- For section 4.1 I used https://www.princeton.edu/~otorres/Regression101.pdf. These slides also helped formatting the output.