**Google Colab Link:** <https://colab.research.google.com/drive/1SRI2Bk4H98rVBY5MUY5O26z-4EDmlV5S?usp=sharing>

CS 412  
SPRING 2022-2023

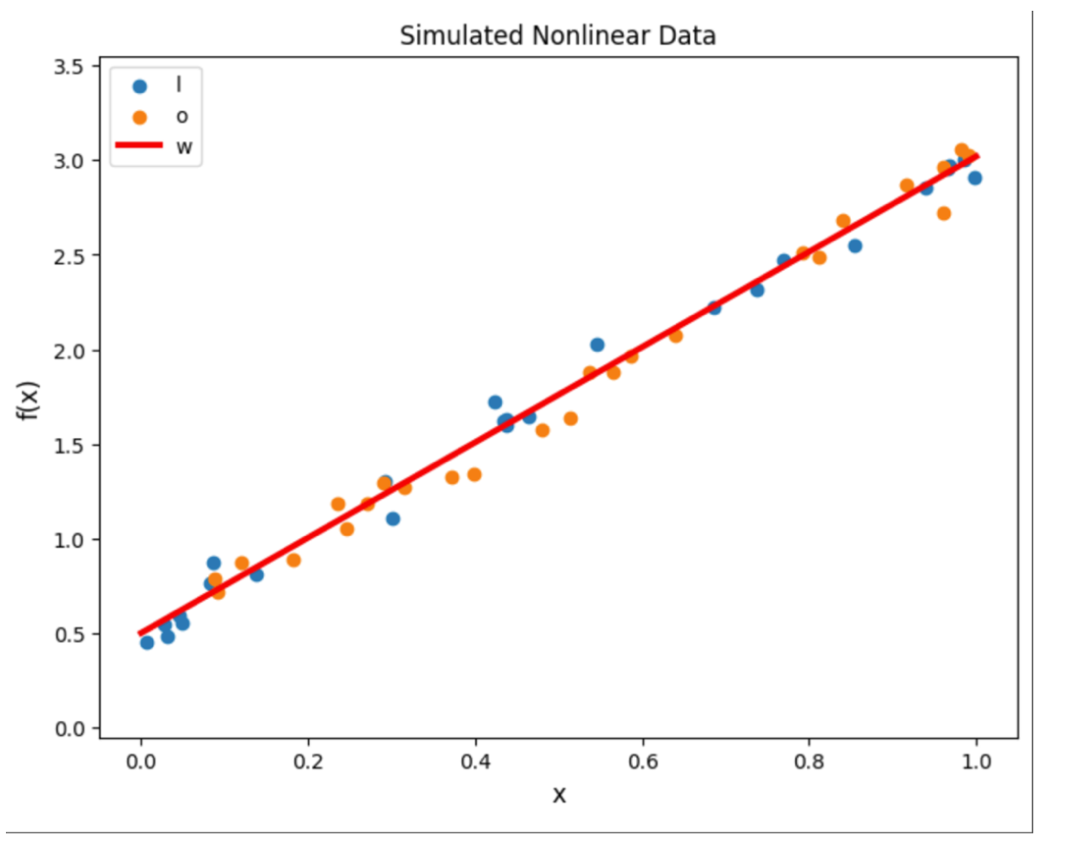
HOMEWORK-2 REPORT

Zeynep Kurtulus

29045

**Part 1.a**

In this part, we perform linear regression using scikit-learn library in Python. We first import LinearRegression class from the sklearn.linear\_model and mean\_squared\_error from sklearn.metrics. Moving on, we initialize LinearRegression Model the we fit the model to the training data and then we use the predict function to find the model’s predictions on the validation set. Moving on, use the mean squared error on the validation set to evaluate the model’s performance on the validation set. In part 1.a the mean squared error of sklearn model is 0.007113027225930444. Finally, we make a scatter plot of the data using plot\_samples function using both train and validation sets, then we use the predict function on x\_grid to find y\_grid and finally display the scatter and the regression line. Below you may see the plot:

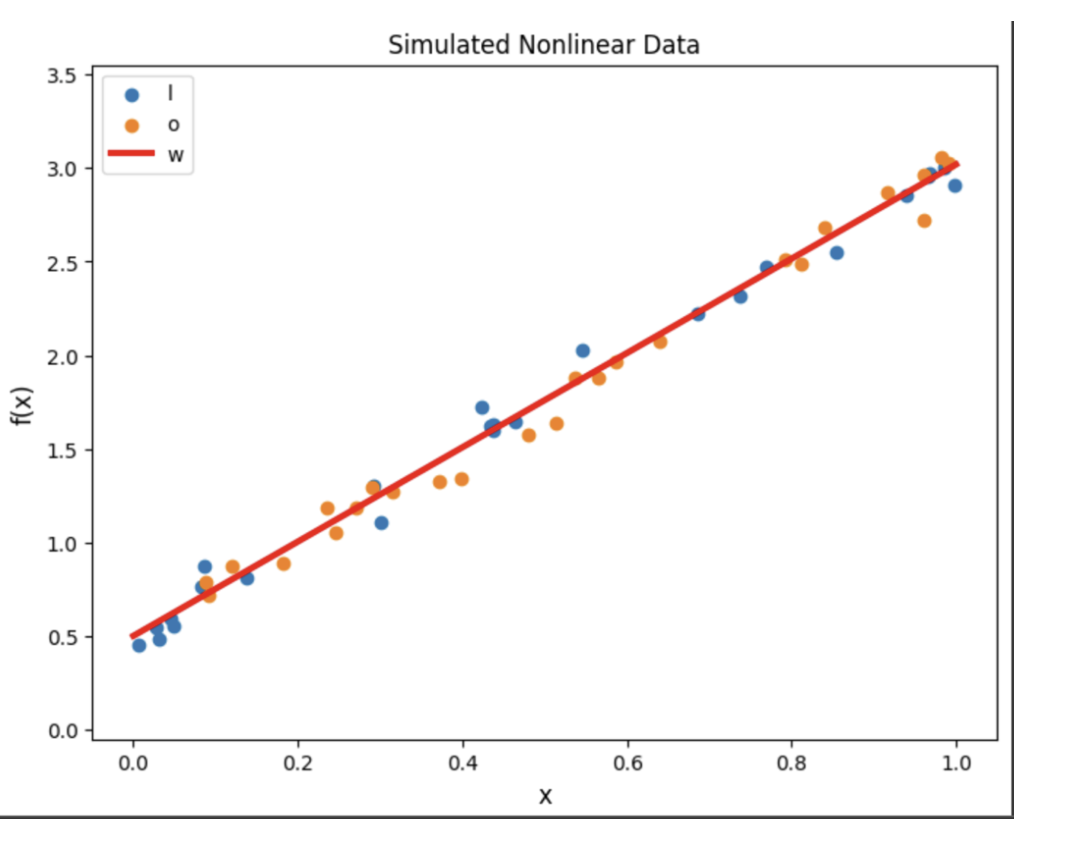


**Part 1.b**

The first line imports the pinv function from the numpy.linalg module. Following two lines create an extended data matrix of the training set (X\_extended\_train) and the validation set (X\_extended\_val) by adding a column of ones to the original data matrices x\_train, x\_valid with concatenate function. Moving on, we find the pseudoinverse of the extended data matrix for the training set with the help of the pinv function. Pseudoinverse is a generalization of the matrix inverse that can be used to solve linear systems that has n unique solution; if the number of data points are less than the number of features the system of linear equations has no unique solution. Then we perform matrix multiplication between the pseudoinverse of the extended data matrix and the target values for the

training set to obtain the regression coefficients w, Below you may see the values for w, regression coefficients:

w is [[1.03127306][1.44122264]]

After we find the models prediction on validation set. Finally, we calculate the mean squared error to evaluate the model’s performance on the validation set. In part 1.b the mean squared error . In part 1.a the mean squared error of sklearn model is 0.007113027225930445. In the final cell of the part 1.b we use plot\_samples function to make a scatter plot, then construct the extended version of x\_grid and finally find the model’s predictions using the regression coefficients. Below you may see the scatter plot and the regression line:

The gradient descent solution is the same in part a and part b

**Part 1.c**

In this part we first define the number of iterations M as 1000 and the learning rate lr as 0.1. Then we initialize the regression coefficients w using np.random.randn function. For each i in range M; at each iteration we calculate the predicted values of y using the current regression coefficients w, below you may see the values for w, regression coefficient:

w is [[0.23887887][0.60580636]]

Then, we calculate the error between the predicted values of y and the true values of y. Moving on, we calculate the gradient of the cost function with respect to variable w, we update the regression coefficients by using the gradient descent algorithm which is w = w - lr \* w\_grad. Again, we calculate the mean squared error of the model on training set. The if part after the MSE does the following; it prints the value of the mean squared error (MSE) of the model on the training set after every 100 iterations, starting from the first iteration (i=0) until the last iteration (i=M-1). The if statement checks if the current iteration is either the first iteration or a multiple of 100 ((i+1) % 100 == 0). If this condition is true, the code inside the block is executed, which prints the MSE on the training set at the current iteration. The MSE values printed are the following :

MSE error at step 1: 2.0150

MSE error at step 100: 0.1729

MSE error at step 200: 0.0684

MSE error at step 300: 0.0294

MSE error at step 400: 0.0148

MSE error at step 500: 0.0094

MSE error at step 600: 0.0074

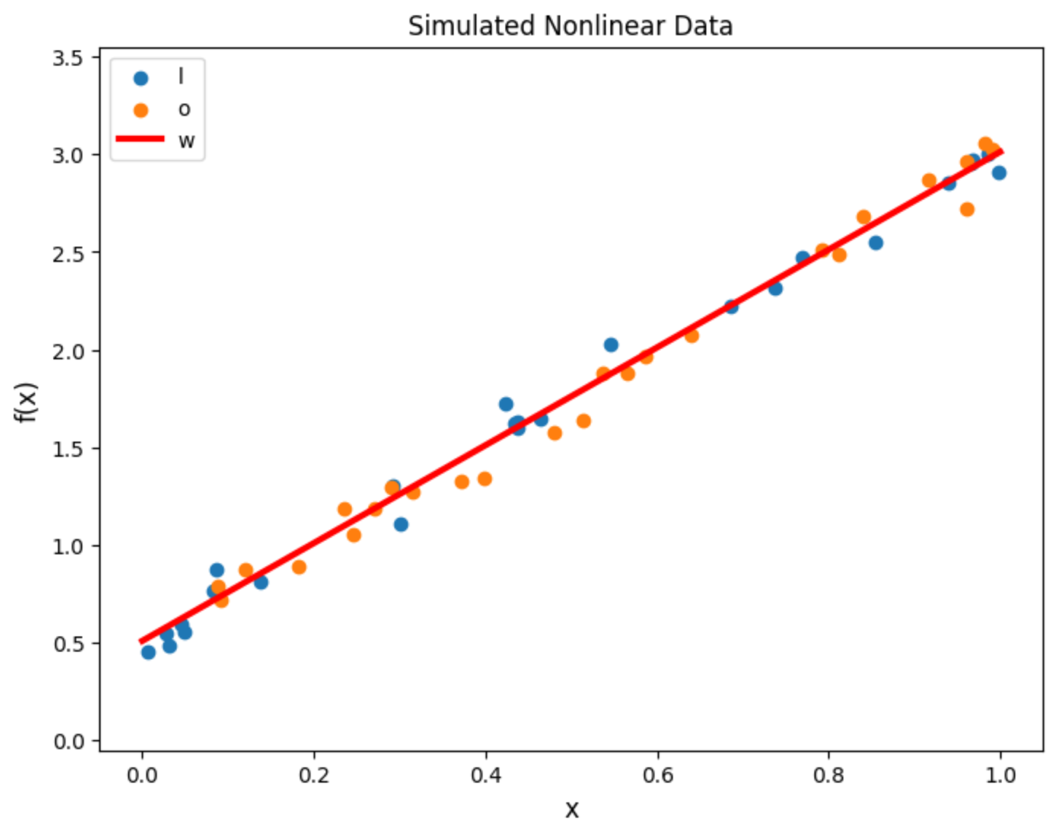
MSE error at step 700: 0.0066

MSE error at step 800: 0.0063

MSE error at step 900: 0.0062

MSE error at step 1000: 0.0062

After this part we use plot\_samples function to print training and validation sets using the plot\_samples() function, then we construct the extended version of x\_grid and finally using the regression coefficients, we find the model's predictions (y\_grid = Xw). Below you may see the plot

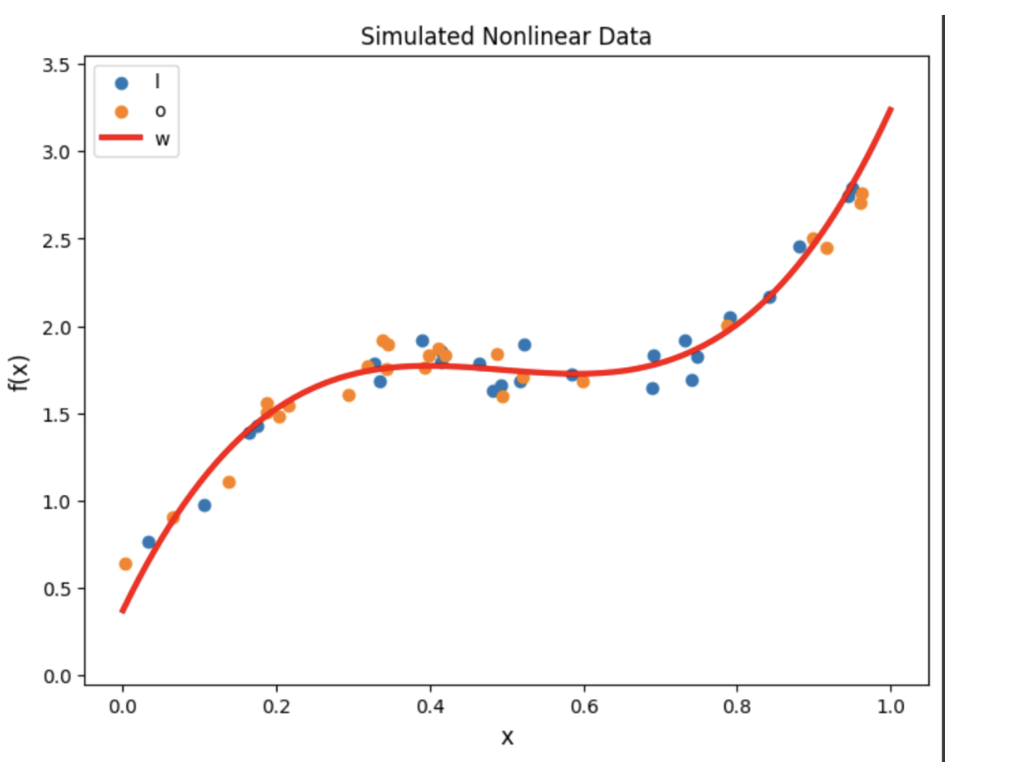


**Part 2.a**

To begin with, we load the data from files using the numpy load() function and assign them to the x and y variables, then we split the data into training and validation sets using the train\_test\_split() function. Moving on, we create a PolynomialFeatures object from the sklearn.preprocessing module, specifying the degree of the polynomial features to be constructed after that, we apply the fit\_transform() function to the training data to create polynomial features and assigns them to the X\_train\_poly variable. Then applies the object's transform() function to the validation data to create polynomial features and assigns them to the X\_val\_poly variable. Moving on, we create a LinearRegression object and use the fit function on the training data to fit the model, then use the predict() function to the validation data to obtain predicted values for y, and calculate the mean squared error (MSE) of the predictions using the mean\_squared\_error() function from the sklearn.metrics. Below you may see the value for MSE:

MSE of sklearn model: 0.01162742864857082

Lastly, we create a scatter plot of the training and validation data using the plot\_samples() function and construct a grid of x values using the np.linspace() function.  
Moving on, we use fit\_transform() function to the grid of x values to create polynomial features and assigns them to the poly\_x\_grid variable, after that we use predict() function to the transformed x grid to obtain predicted values for y and assigns them to the y\_grid variable. Below you may see the scatter plot:



The regression coefficients w values used are:

w values: [[0.21059014][0.12093332]]

In this part below you may see the results of the MSE value if change the value of the order variable:

For order: 1

MSE of sklearn model: 0.054584504330446676

For order: 3

MSE of sklearn model: 0.01162742864857082

For order: 5

MSE of sklearn model: 0.006837024947577867

For order: 7

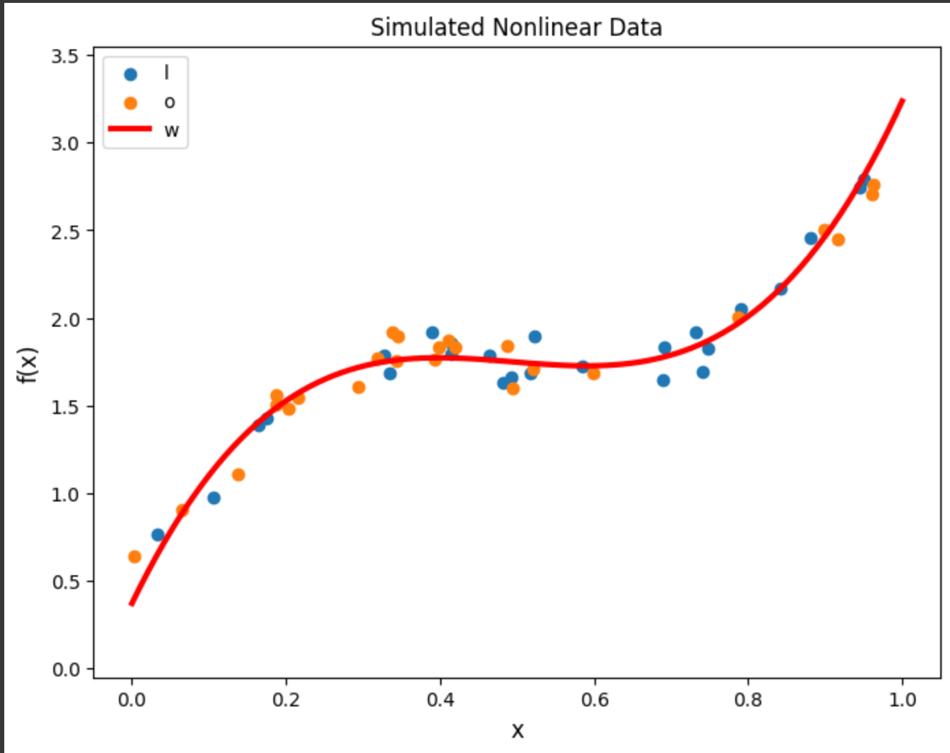
MSE of sklearn model: 0.01029019267681711

Order variable specifies the degree of the polynomial features that will be created from the input data. For example, if order is set to 3, then the PolynomialFeatures object will create polynomial features of degree 0, 1, 2, and 3 from the original input features. These polynomial features will then be used to fit the LinearRegression model, which will make predictions on the transformed data. The higher the order of the polynomial features, the more complex the resulting model will be, which may lead to overfitting if the data is limited or noisy, this occurs when the model fits the noise in the training data instead of the underlying pattern, leading to poor generalization to new, unseen data. When the order variable is low, it means that the polynomial features used in the model are of lower degree. This can lead to a simpler model that may not capture complex relationships between the input features and the target variable. the model may underfit the training data, meaning that it is not able to capture the underlying pattern and is too simple to fit the data well.That is why to kind of a chose a middle value I have order = 3 in my colab file.

**Part 2.b**

In this part, we first load the data (x and y) from two separate .npy files using np.load() function, then we split the data into training and validation sets using train\_test\_split() function from sklearn. Then we create a PolynomialFeatures object using degree 3 (selected in part 2.a) and use fit\_transform() function fit and transform training data to polynomial features. Moving on, we Use transform() function of PolynomialFeatures object to transform validation data to polynomial features and initialize a LinearRegression model. Then we use fit() function of LinearRegression model to fit the model to the polynomial features of training data and use the predict() function to predict the dependent variable for validation data. Lastly, Calculate the mean squared error (MSE) of the model on validation data using mean\_squared\_error() function. Below you may see the MSE value:

MSE of our model: 0.011627428648570914

Then we plot the training and validation data using plot\_samples() function and create a grid of x values to use in plotting the model's predictions. We transform the x\_grid using PolynomialFeatures object to polynomial features and we usethe regression coefficients to find the model's predictions for y values on x\_grid. Finally we plot the model's predictions on the x\_grid using plot() function. You may see the scatter plot below: