



## Table of Contents

### Lab 01: Linear regression

1. The hypothesis set
2. Performance measure and the learning goal
3. Implementation
  - Import library
  - Create data
  - Visualize data

### Selection of the model

- Train our model and visualize
4. Polinomial regression
  - Abstract the problem
  - Solve the problem in code

```
1 begin
2   using PlutoUI # visualization purpose
3   TableOfContents(title="📄 Table of Contents", indent=true, depth=3, aside=true)
4 end
```

## Lab 01: Linear regression

Copyright © Department of Computer Science, University of Science, Vietnam National University, Ho Chi Minh City

- Student name: Hoang Anh Tra
- ID: 21127453

### How to do your homework

- You will work directly on this notebook; the word **TODO** indicates the parts you need to do.
- You can discuss the ideas as well as refer to the documents, but *the code and work must be yours*.

### How to submit your homework

- Before submitting, save this file as <ID>.jl. For example, if your ID is 123456, then your file will be 123456.jl. And export to PDF with name 123456.pdf then submit zipped source code and pdf into 123456.zip onto Moodle.

#### Note

**Note that you will get 0 point for the wrong submit.**

### Content of the assignment:

- Linear regression
- Polinomial linear regression

# 1. The hypothesis set

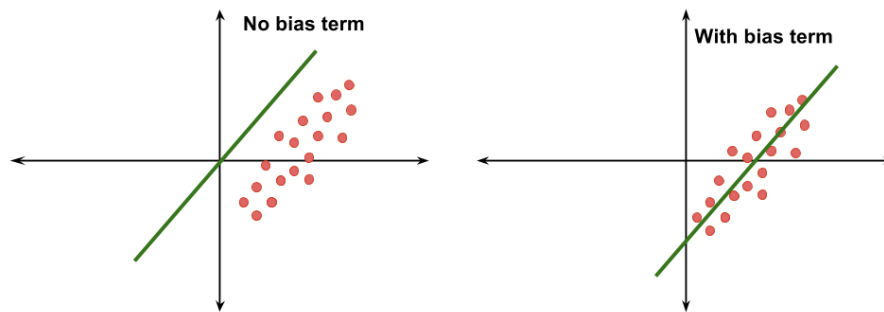
- Linear regression is a linear model, e.g. a model that assumes a linear relationship between the input variables ( $x$ ) and the single output variable ( $y$ ). More specifically, that  $y$  can be calculated from a linear combination of the input variables ( $x$ ).
- Generally, a linear model will make predictions by calculating a weighted sum of the input features (independent variables).

$$\hat{y} = w_0 + \sum_{i=1}^d w_i x_i$$

- $\hat{y}$  is the predicted value.
  - $d$  is the number of features.
  - $x_i$  is the  $i^{th}$  feature value.
  - $w_j$  is the  $j^{th}$  model parameter (including the bias term  $w_0$  and the feature weights  $w_1, w_2, \dots, w_d$ )
- You can rewrite the first equation in the matrix form as following:

$$\hat{y} = h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \cdot \mathbf{x}$$

- $\mathbf{w}$  is the model **parameter vector** (including the bias term  $w_0$  and the feature weights  $w_1, w_2, \dots, w_n$ ).
- $\mathbf{w}^T$  is a transpose of  $\mathbf{w}$  (a row vector instead of column vector).
- $\mathbf{x}$  is the instance's **feature vector**, containing  $x_0$  to  $x_d$ , with  $x_0$  always equal to 1.
- $\mathbf{w}^T \cdot \mathbf{x}$  is the dot product of  $\mathbf{w}^T$  and  $\mathbf{x}$ .
- $h_{\mathbf{w}}$  is the hypothesis function, using the parameters  $\mathbf{w}$ .



## 2. Performance measure and the learning goal

- Before we start to train the model, we need to determine how good the model fits the training data. There are a couple of ways to determine the level of quality, but we are going to use the most popular one and that is the **MSE** (Mean Square Error). We need to find the value for  $\mathbf{w}$  that will minimize the MSE:

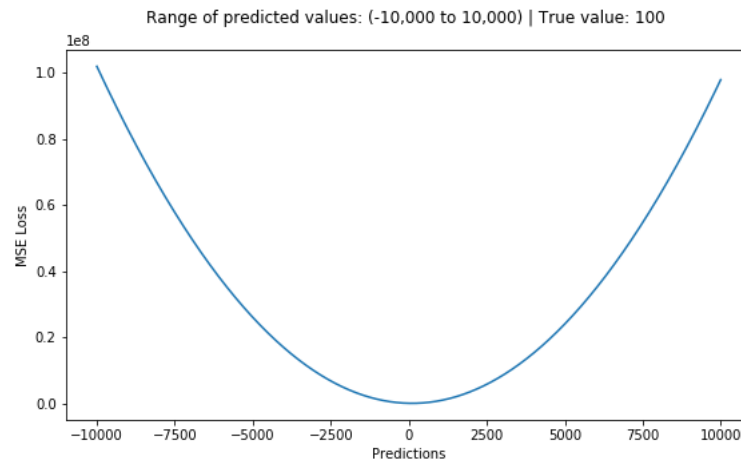
$$\mathbf{w} = \arg \min MSE_{\mathcal{D}_{train}}$$

- MSE on the train set  $\mathcal{D}_{train}$  denoted as  $(\mathbf{X}, \mathbf{y})$  including  $n$  samples  $\{(\mathbf{x}_1, y_1), \dots (\mathbf{x}_n, y_n)\}$

$$MSE(\mathbf{X}, h_{\mathbf{w}}) = \frac{1}{m} \sum_{i=1}^m (\mathbf{w}^T \cdot \mathbf{x}_i - y_i)^2$$

$$MSE(\mathbf{X}, h_{\mathbf{w}}) = \frac{1}{m} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

- Example below is a plot of an MSE function where the true target value is 100, and the predicted values range between -10,000 to 10,000. The MSE loss (Y-axis) reaches its minimum value at prediction (X-axis) = 100. The range is 0 to  $\infty$ .



- To find the value of  $\mathbf{w}$  that minimizes the MSE cost function the most common way (*we have known since high school*) is to solve the derivative (gradient) equation.

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

- $\hat{\mathbf{w}}$  is the value of  $\mathbf{w}$  that minimizes the cost function
- In order to reduce the complexity of this approach, I will provide you another version of computing the optimal  $\mathbf{w}$ :

$$\mathbf{X}\mathbf{w} = \mathbf{y}$$

$$\Leftrightarrow \mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{X}^T \mathbf{y}$$

$$\Leftrightarrow \mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

## 3. Implementation

### Import library

```
PlotlyBackend()
```

```
1 begin
2   using Plots, Distributions
3   plotly()
4 end
```

For saving to png with the 'Plotly' backend 'PlotlyBase' and 'PlotlyKaleido' need to be installed.

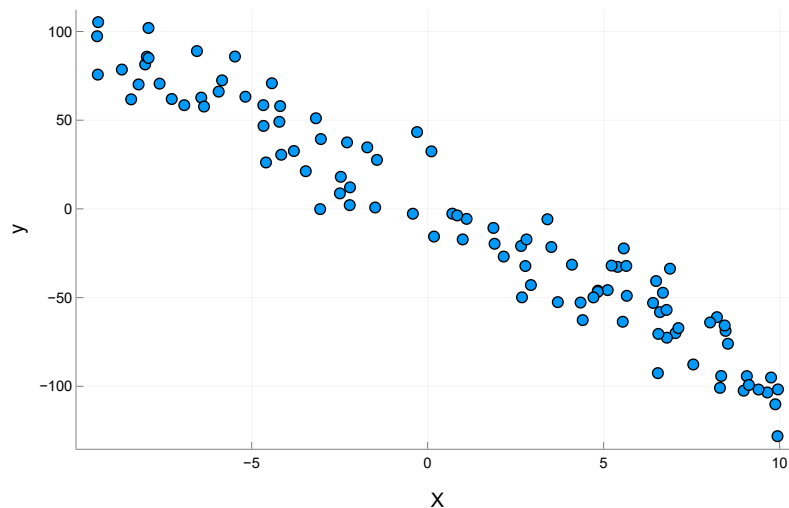
```
err: ArgumentError("Package PlotlyBase not found in cur" ... 52 bytes ... " to instal
```

## Create data

```
([-8.02623, -1.71562, -5.47458, 1.86865, -3.05296, 4.40586, -9.39089, 9.06904, 7.04082,
```

```
1 begin
2   function create_data(f, num_data=100)
3       # f is the function that is used to generate data
4       X = Base.rand(Distributions.Uniform(-10,10), num_data)
5       y = f.(X);
6       return X, y
7   end
8
9   a = Base.rand(Distributions.Uniform(-10,10), 1)[1]
10  f(x) = a*x + Base.rand(Distributions.Normal(1,15), 1)[1]
11  X, y = create_data(f)
12 end
```

## Visualize data



### TODO:

My observations about data:

- The data is roughly evenly distributed around the X-axis.
- There is a positive correlation between X and Y, meaning that as X increases, Y tends to increase as well.
- The relationship between X and Y is not perfectly linear, but there is a slight upward trend.
- There are a few outliers, meaning that there are a few data points that are significantly different from the rest of the data.

## Training function

train\_linear\_regression (generic function with 1 method)

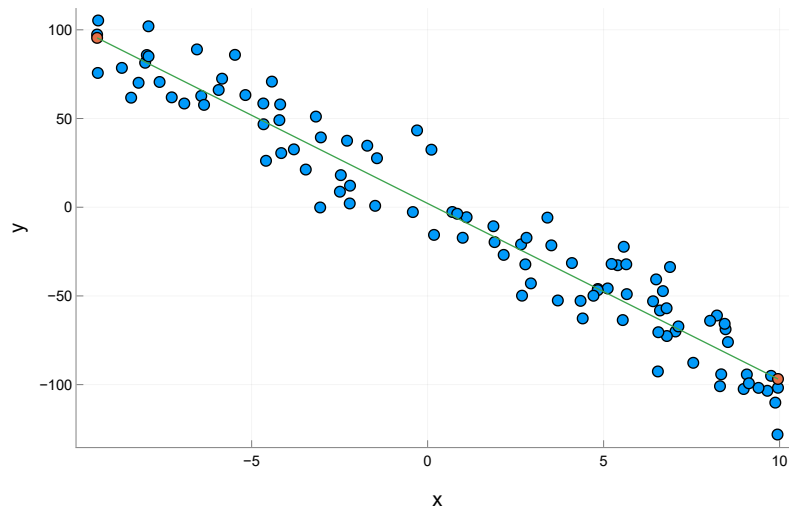
```
1 function train_linear_regression(X, y)
2     """
3     Trains Linear Regression on the dataset (X, y).
4
5     Parameters
6     -----
7     X : Matrix, shape (n, d + 1)
8         The matrix of input vectors (each row corresponds to an input vector);
9         the first column of this matrix is all ones (corresponding to x_0).
10    y : Matrix, shape (n, 1)
11        The vector of outputs.
12
13    Returns
14    -----
15    w : Matrix, shape (d + 1, 1)
16        The vector of parameters of Linear Regression after training.
17    """
18    # TODO: compute your weight
19
20    w = inv(transpose(X) * X) * transpose(X) * y
21
22    return w
23 end
```

```
1 begin
2     # Construct one_added_X
3     # TODO: First column of one_added_X is all ones (corresponding to x_0).
4
5
6     # Concatenate the vector of ones to the front of the matrix X.
7     one_added_X = [transpose(ones(1, size(X)[1])) X]
8
9     println("size of one_added_X = ", Base.size(one_added_X))
10    println("size of y = ", Base.size(y))
11 end
```

```
size of one_added_X = (100, 2)
size of y = (100,)
```



## Train our model and visualize



```

1 begin
2   w = train_linear_regression(one_added_X, y)
3
4   # Visualize result
5   predicted_ys = one_added_X*w
6   scatter(X, y, xlabel="x", ylabel="y", legend=false)
7   x_min = minimum(X)
8   x_max = maximum(X)
9   xs = [x_min x_max]'
10
11   # Construct ones_added_xs
12   # TODO: First column of ones_added_xs is all ones (corresponding to x_0).
13
14   ones_added_xs = [transpose(ones(1, size(xs)[1])) xs]
15
16   predicted_ys = ones_added_xs*w
17   scatter!(xs, predicted_ys, legend=false)
18   plot!(xs, predicted_ys, legend=false)
19
20 end

```

## 4. Polynomial regression

- Observe the following dataset. Can we use linear regression model to fit this data?

We should use a polynomial regression model instead of a linear regression model to fit the dataset you sent because the relationship between the independent and dependent variables in the dataset is non-linear. A linear regression model assumes that the relationship between the independent and dependent variables is linear, meaning that the predicted value of the dependent variable changes linearly with the independent variable. However, the dataset you sent shows a clear non-linear relationship between the independent and dependent variables.

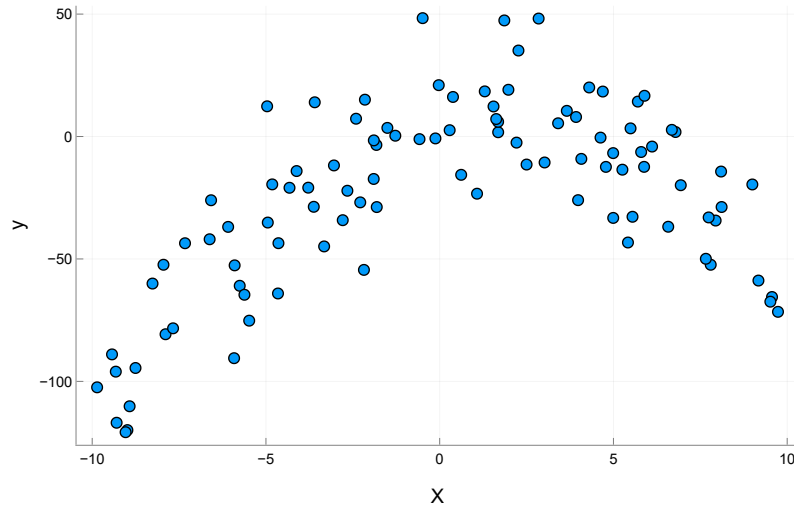
```

1 md"""
2 ## 4. Polynomial regression
3
4 - Observe the following dataset. Can we use linear regression model to fit this data?
5
6 We should use a polynomial regression model instead of a linear regression model to
  fit the dataset you sent because the relationship between the independent and
  dependent variables in the dataset is non-linear.
7 A linear regression model assumes that the relationship between the independent and
  dependent variables is linear, meaning that the predicted value of the dependent
  variable changes linearly with the independent variable. However, the dataset you
  sent shows a clear non-linear relationship between the independent and dependent
  variables.
8 """

```

([5.2577, -2.286, 5.79775, 1.97733, 5.70224, 8.09802, 2.26646, 3.40795, 0.382734, more

```
1 begin
2   a_ = Base.rand(Distributions.Uniform(-5,5))
3   b_ = Base.rand(Distributions.Uniform(-10,10))
4   # c_ = Base.rand(Distributions.Uniform(-10,10))
5   g(x) = a_*x^2 + b_*x + Base.rand(Distributions.Normal(1,20),1)[1]
6   X_, y_ = create_data(g)
7 end
```



## Abstract the problem

- For this kind of datasets, you have to **change the hypothesis set**. For example, in this case, we assume that our data is in the parabol form. Therefore, the hypothesis set will be  $\hat{y} = ax^2 + bx + c$ . Another assumption:  $\hat{y} = ax^3 + bx^2 + cx + d$ .
- In general, we have the polinomial regression form:

$$\hat{y} = w_0 + \sum_{i=1}^d w_i x_i^i$$

- Re-write the equation above:

$$\hat{y} = h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \cdot \mathbf{x}, \text{ where } \mathbf{x} = \begin{bmatrix} x_0 = 1 \\ x_1^1 \\ \vdots \\ x_d^d \end{bmatrix} \in \mathbb{R}^{d+1}$$

- To solve this problem, we have to find  $\mathbf{w}$  such that:

$$\min_{\mathbf{w}} MSE(X, h_{\mathbf{w}}) = \min_{\mathbf{w}} \frac{1}{m} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

- Recall the solution of MSE in section 2:

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

- Now, it's time for coding!

## Solve the problem in code

**Step 1:** Create polynomial feature

$$\mathbf{X} = \{\mathbf{x}_i = (x_0 = 1, x_1^1, x_2^2, \dots, x_d^d)\}_{i=1}^n$$

```

1 # TODO: Create X
2
3 begin
4     function poly_features(X, K)
5         # X: inputs of size N x 1
6         # K: degree of the polynomial
7         # Create the matrix of polynomial features.
8         X = zeros(size(X, 1), K + 1)
9
10        # Fill in the matrix of polynomial features.
11        for i = 0:K
12            X[:, i+1] = X.^i
13        end
14        return X
15    end
16
17    # assume that our data is in form of a parabol
18    X = poly_features(X_, 2)
19    println(size(X))
20 end

```

(100, 3)



**Step 2:** train our model and find  $w$

$w = [4.28896, 2.93761, -0.953639]$

```

1 # TODO: train our model
2
3 w = train_linear_regression(X, y_)

```

**Step 3:** Visualize our result

