Fundamentals of Regression by Machine Learning

My Studying Log of fundamentals about Machine Learning Regression.

Table of contents

Introduction

This is my studying log about machine learning, regression. I referred to a following book.
Pythonで動かして学ぶ! あたらしい機械学習の教科書
I extracted some important points and some related sample python codes and wrote them as memo in this article.

Author

Makoto Ito

Linear model with 1 dimensional input

Input data: Age

$$\boldsymbol{x} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \\ \vdots \\ x_{N-1} \end{pmatrix}$$

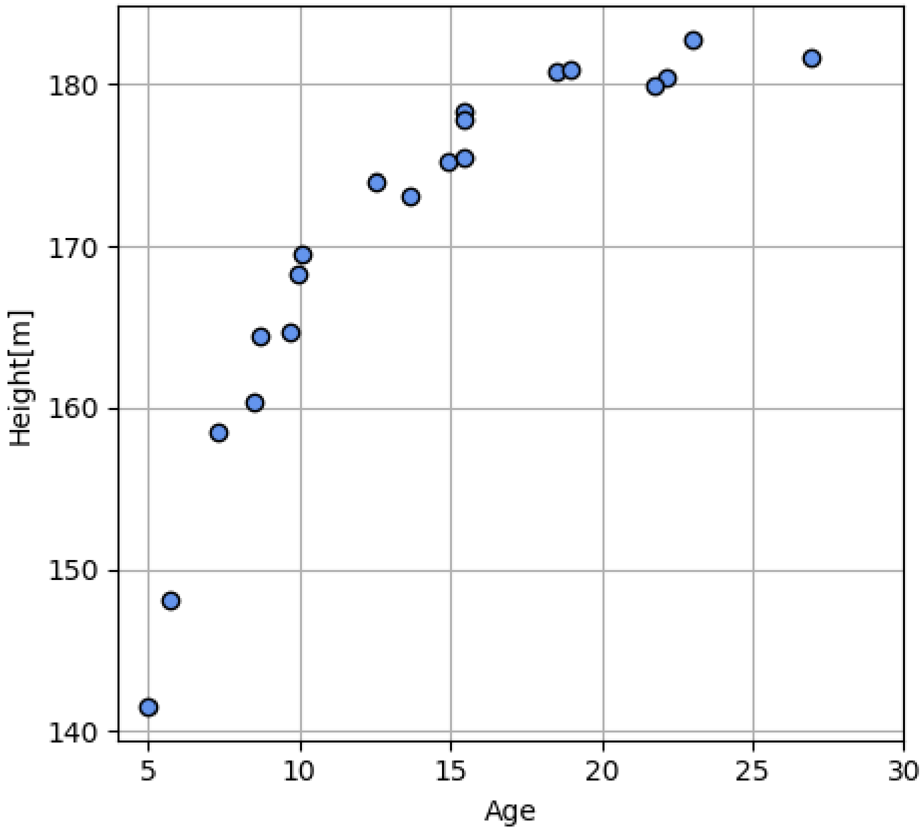
Target data: Height

$$
\boldsymbol{t} = \left(\begin{array}{c} t_0 \\ t_1 \\ \vdots \\ t_n \\ \vdots \\ t_{N-1} \end{array} \right)
$$

N means the number of people and $N = 20$. A purpose of this regression is predicting a height with an age of a person who is not included the databases.

Data generation

This data was generated by generate_1dimensional_linear_data.py



Linear model definition

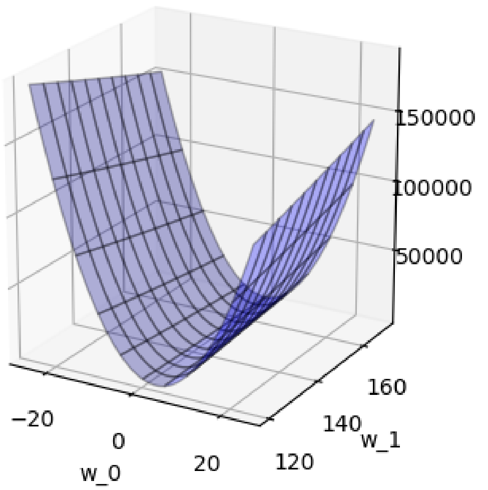
- Linear equation:

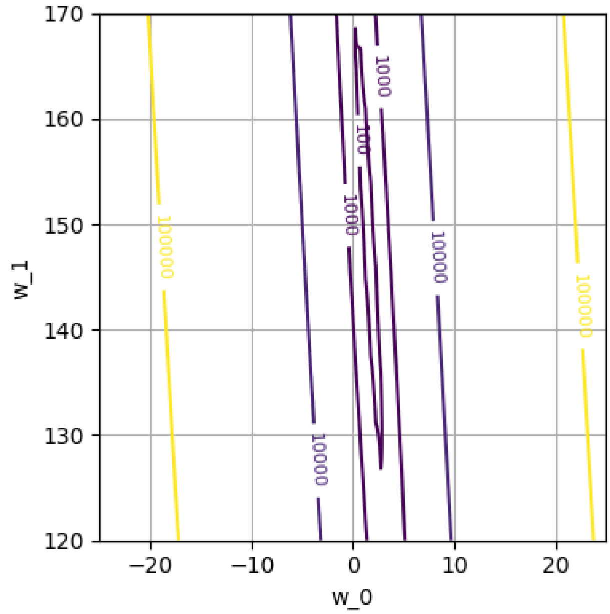
$$
y_n = y(x_n) = w_0 x_n + w_1
$$

- Mean squared error function:

$$
J = \frac{1}{N} \sum_{n=0}^{N-1} (y_n - t_n)^2
$$

- plot relationship between w and J:
  This figure was created by mean_squared_error_function.py





We need to decide w_0 and w_1 which minimize mean squared error, J. Depend on the above graph, J has a shape like a valley. And then, the value of J is changing to the direction of w_0, w_1.
When w_0 is about 3 and w_1 is about 135, J will be minimized.

Gradient method

Gradient method is used for calculating w_0 and w_1 which minimize the value of J. This method rpeat the following calculation:

1. Select a initial point, w_0 and w_1 on the valley of J.
2. calculate a gradient at the selected point.
3. w_0 and w_1 are moved to the direction which the value of J most decline.
4. Finally, w_0 and w_1 will reach values which minimize the value of J.

- Gradient to the going up direction:

$$\nabla_{wJ} = \begin{bmatrix} \frac{\delta J}{\delta w_0} \\ \frac{\delta J}{\delta w_1} \end{bmatrix} = \begin{bmatrix} \frac{2}{N}\sum_{n=0}^{N-1}(y_n - t_n)x_n \\ \frac{2}{N}\sum_{n=0}^{N-1}(y_n - t_n) \end{bmatrix}$$

- Gradient to the going down direction:

$$\nabla_{wJ} = -\begin{bmatrix} \frac{\delta J}{\delta w_0} \\ \frac{\delta J}{\delta w_1} \end{bmatrix} = \begin{bmatrix} -\frac{2}{N}\sum_{n=0}^{N-1}(y_n - t_n)x_n \\ -\frac{2}{N}\sum_{n=0}^{N-1}(y_n - t_n) \end{bmatrix}$$

- Learning algorithm:

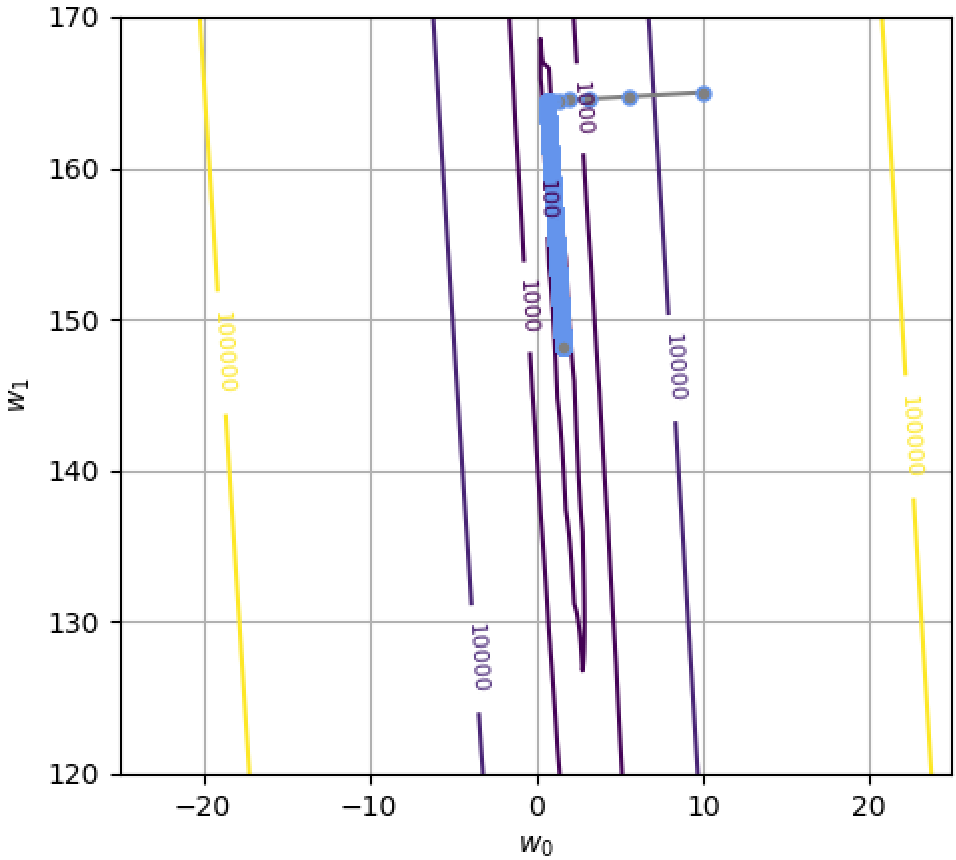
$$w(t+1) = w(t) - \alpha \nabla_{wJ}|_{w(t)}$$

α is a positive number and called "Learning rate" which can adjust a update width of w. The bigger this number is, the bigger the update width is, but a convergence of calculation will be unstable.

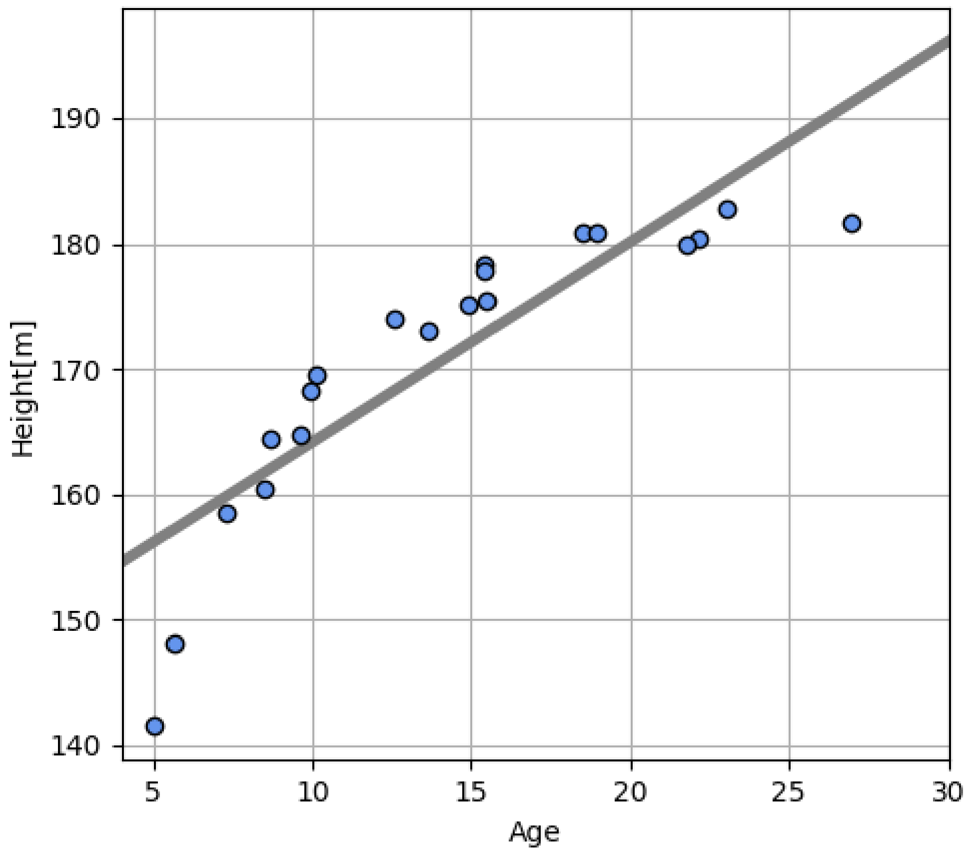
Learning Result

This learning was executed by gradient_method.py

- Learning behavior plot:
  Initial point: [10.0, 165.0]
  Final point: [1.598, 148.172]
  Number of iteration: 12804



- Predicted linear line plot:
  Mean squared error: 29.936629[cm^2]
  Standard deviation: 5.471[cm]



Point to notice

The result which is solved by Gradient method is just a local minimum value and not always global minimum value. Practically, we need to try gradient method with various initial values and select the minimum result value.

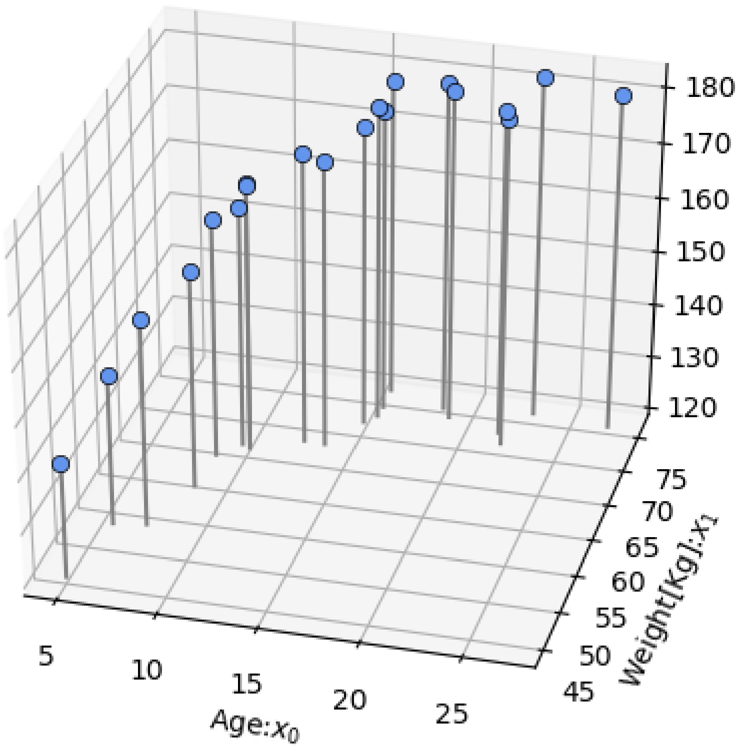
Plane model with 2 dimensional input

In this case, data vector x is extended to 2 dimensional data, $x = (x_0, x_1)$. x_0 is age and x_1 is weight.

Data generation

This data was generated by generate_2dimensional_plane_data.py

$$Weight = 23 \times \frac{Height^2}{100} + Noise$$



Plane model

- Definition of surface function:

$$y(x) = w_0x_0 + w_1x_1 + w_2$$

- Mean squared error function:

$$J = \frac{1}{N}\sum_{n=0}^{N-1}(y(x_n) - t_n)^2 = \frac{1}{N}\sum_{n=0}^{N-1}(w_0x_{n,0} + w_1x_{n,1} + w_2 - t_n)^2$$

- Gradient:

$$\frac{\sigma J}{\sigma w_0} = 0, \frac{\sigma J}{\sigma w_1} = 0, \frac{\sigma J}{\sigma w_2} = 0$$

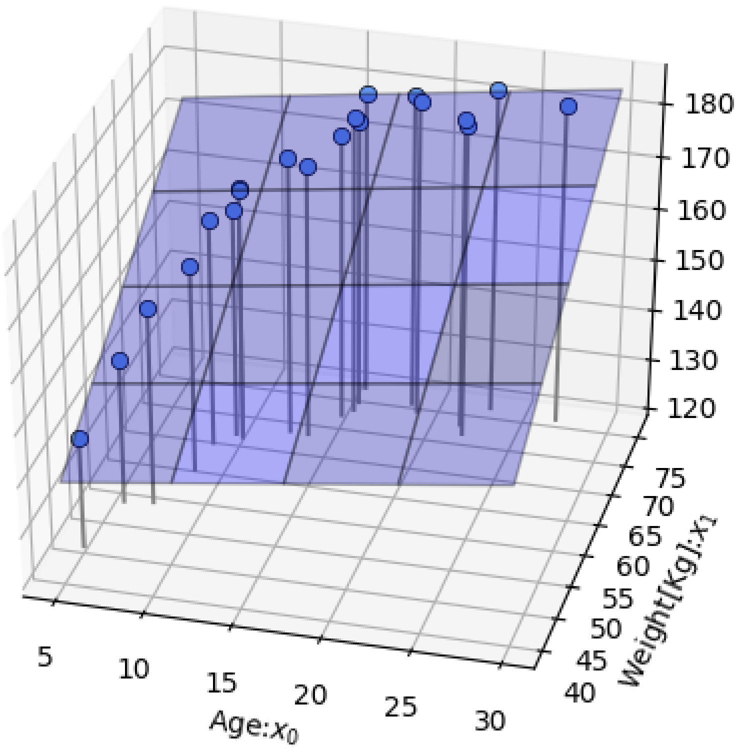
- Optimal parameters:

$$w_0 = \frac{cov(t, x_1)cov(x_0, x_1) - var(x_1)cov(t, x_0)}{cov(x_0, x_1)^2 - var(x_0)var(x_1)}$$

$$w_1 = \frac{cov(t, x_0)cov(x_0, x_1) - var(x_0)cov(t, x_1)}{cov(x_0, x_1)^2 - var(x_0)var(x_1)}$$

$$w_2 = -w_0 \frac{1}{N} \sum_{n=0}^{N-1} x_{n,0} - w_1 \frac{1}{N} \sum_{n=0}^{N-1} x_{n,1} + \frac{1}{N} \sum_{n=0}^{N-1} t_n$$

- Learning result:
  This learning was executed by learning_plane_model.py
  $w_0 = 0.4$, $w_1 = 1.0$, $w_2 = 95.5$
  Standard deviation: 2.374[cm]



D-dimensional Linear Regression Model

- It requires a lot of work to derive all of formulas at different dimension. So, we need to define the number of dimension as a variable, D.

$$y(x) = w_0 x_0 + w_1 x_1 + \cdots + w_{D-1} x_{D-1} + w_D$$

- We can shorten the above model with Matrix representation.

$$y(x) = [w_0 \cdots w_{D-1}] \begin{bmatrix} x_0 \\ \vdots \\ x_{D-1} \end{bmatrix} = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}$$

Solution of model

$$J(\boldsymbol{w}) = \frac{1}{N}\sum_{n=0}^{N-1}(y(x_n) - t_n)^2 = \frac{1}{N}\sum_{n=0}^{N-1}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n - t_n)^2$$

$$\frac{\partial J}{\partial w_i} = \frac{1}{N}\sum_{n=0}^{N-1}\frac{\partial}{\partial w_i}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n - t_n)^2 = \frac{2}{N}\sum_{n=0}^{N-1}\frac{\partial}{\partial w_i}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n - t_n)x_{n,i}$$

$$\sum_{n=0}^{N-1}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n - t_n)x_{n,i} = 0$$

$$\sum_{n=0}^{N-1}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n - t_n)[x_{n,0}, x_{n,1}, \cdots, x_{n,D-1}] = \sum_{n=0}^{N-1}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n - t_n)\boldsymbol{x}_{\boldsymbol{n}}^{\mathrm{T}} = [0\ 0\ \cdots\ 0]$$

$$\sum_{n=0}^{N-1}x_n x_n^{\mathrm{T}} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\,, \sum_{n=0}^{N-1}x_n x_n^{\mathrm{T}} = \boldsymbol{t}^{\mathrm{T}}\boldsymbol{X}$$

$$(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} - \boldsymbol{t}^{\mathrm{T}}\boldsymbol{X})^{\mathrm{T}} = [0\ 0\ \cdots\ 0]^{\mathrm{T}}$$

$$\boldsymbol{w} = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{t}$$

The right side of the above formula, $(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}$ is called "Moore-Penrose Pseudo-inverse matrix".

Extension to plane not passing through origin

Vector $\boldsymbol{x}$ can be thought as 3 dimensional vector by adding 3rd element which is always "1". In case that $x_2 = 1$,

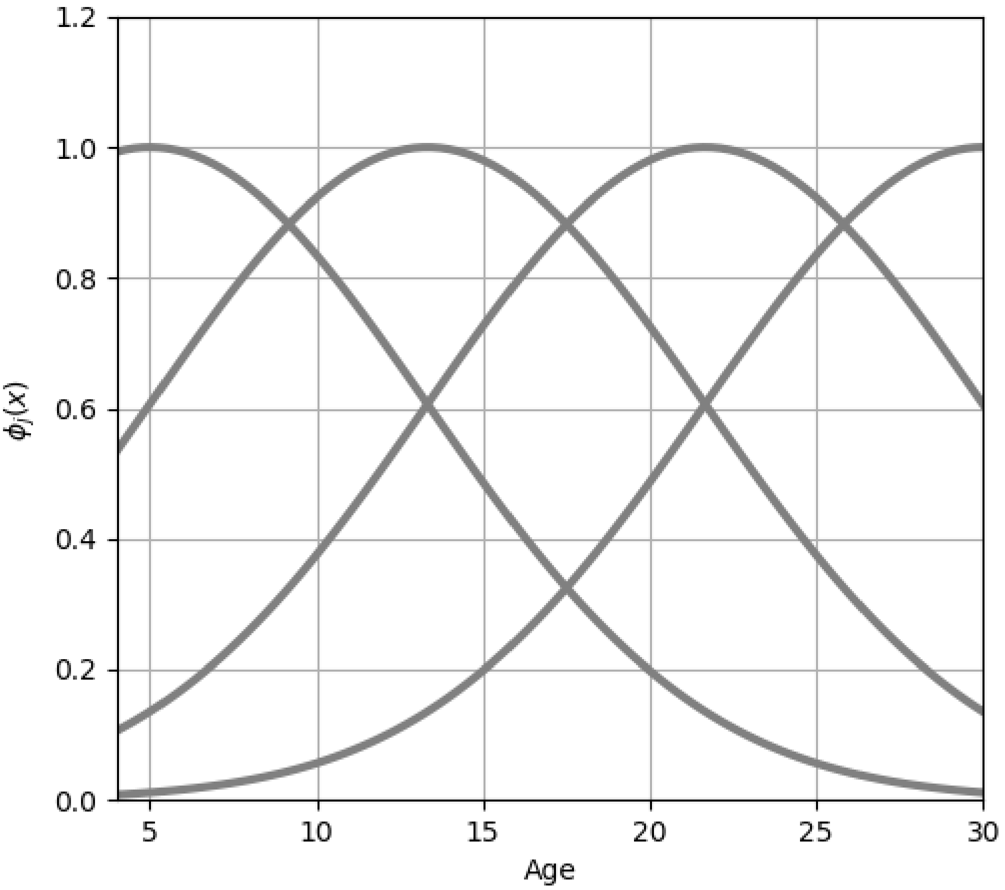
$$y(\boldsymbol{x}) = w_0 x_0 + w_1 x_1 + w_2 x_2 = w_0 x_0 + w_1 x_1 + w_2$$

Linear basis function

- Way of thinking
  x of Linear Regression model is replaced with Basis function $\phi(x)$ to create a function which has various shapes.
- Gaussian function
  Gaussian function is used as basis function in this section. Basis function is used as multiple sets and a suffix j is attached in the formula. s is a parameter to adjust a spread of the function.

$$\phi_j(x) = exp\{-\frac{(x-\mu_j)^2}{2s^2}\}$$

- Combined function of M gaussian functions
  This figure is created gaussian_basis_function.py
  In order from left, $\phi_0(x)$, $\phi_1(x)$, $\phi_2(x)$, $\phi_3(x)$.



M is the number of combined functions. In the above, $M = 4$
Weight parameters for each function: w_0, w_1, w_2, w_3
A parameter for adjusting up and down movement of model: w_4
w_4 is for a dummy function, $\phi_4(x) = 1$.

$$y(x, \boldsymbol{w}) = w_0\phi_0(x) + w_1\phi_1(x) + w_2\phi_2(x) + w_3\phi_3(x) + w_4$$
$$y(\boldsymbol{x}, \boldsymbol{w}) = \sum_{j=0}^{M} w_j\phi_j(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$$

- Mean squared error J

$$J(\boldsymbol{w}) = \frac{1}{N}\sum_{n=0}^{N-1}\{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x_n}) - t_n\}^2$$

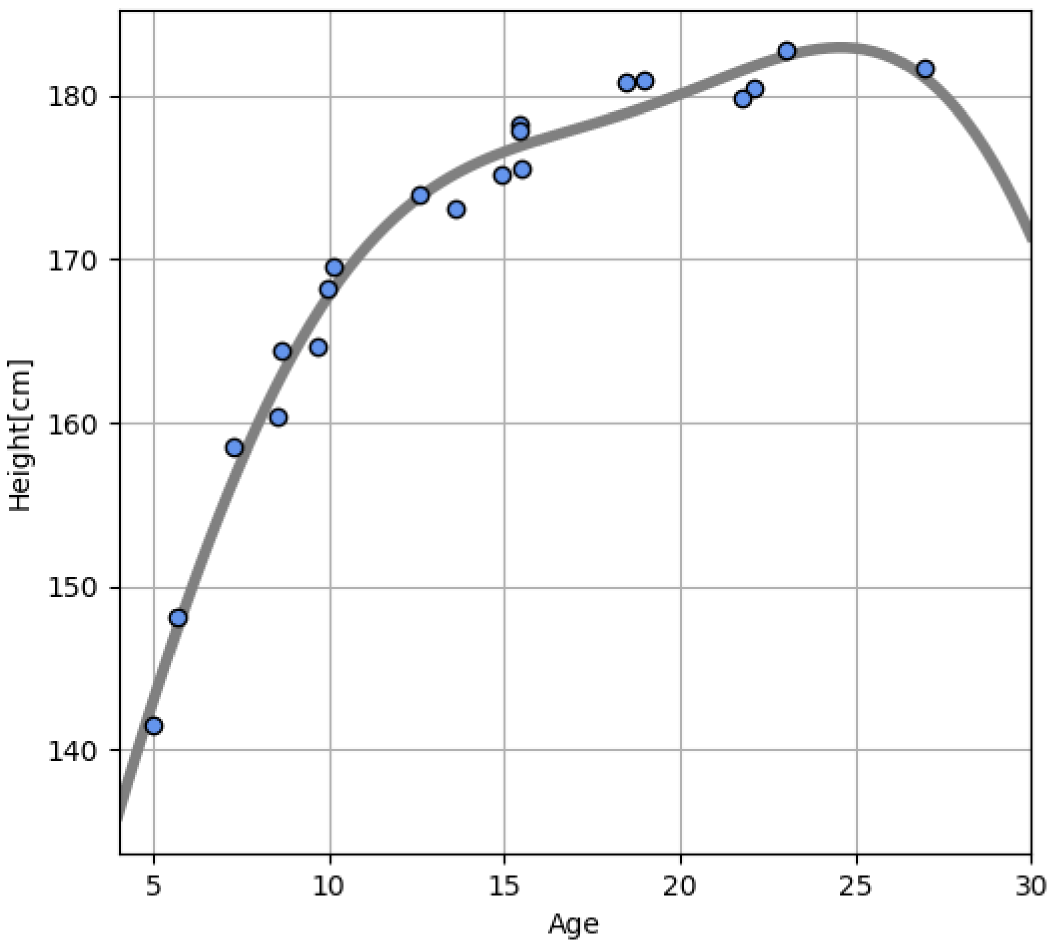
- Solution $\boldsymbol{w}$

$$\boldsymbol{w} = (\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{\phi})^{-1}\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{t}$$

- Preprocessed input data $\boldsymbol{\phi}$
  $\boldsymbol{\phi}$ is called "Design matrix".

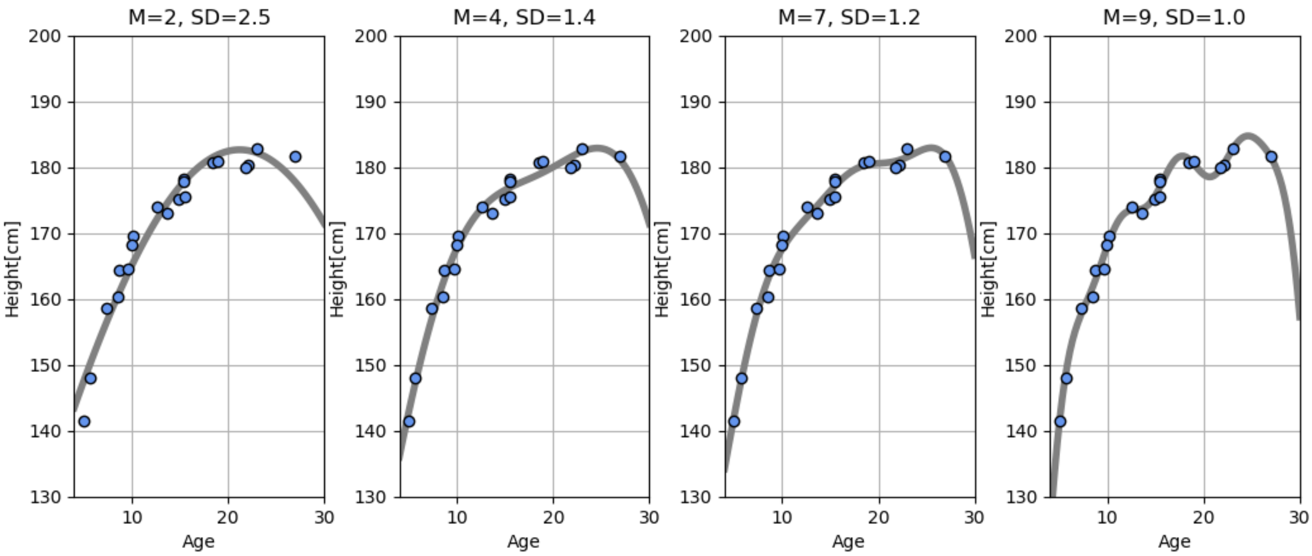
$$\boldsymbol{\phi} = \begin{bmatrix} \phi_0(\boldsymbol{x}_0) & \phi_1(\boldsymbol{x}_0) & \cdots & \phi_M(\boldsymbol{x}_0) \\ \phi_0(\boldsymbol{x}_1) & \phi_1(\boldsymbol{x}_1) & \cdots & \phi_M(\boldsymbol{x}_1) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_0(\boldsymbol{x}_{N-1}) & \phi_1(\boldsymbol{x}_{N-1}) & \cdots & \phi_M(\boldsymbol{x}_{N-1}) \end{bmatrix}$$

- Learning Result
  This learning was executed by learning_gaussian_function.py
  $w_0 = 62.2$, $w_1 = 71.8$, $w_2 = 30.4$, $w_3 = 110.6$, $w_4 = 31.9$
  Standard deviation: 1.43[cm]



Overfitting problem

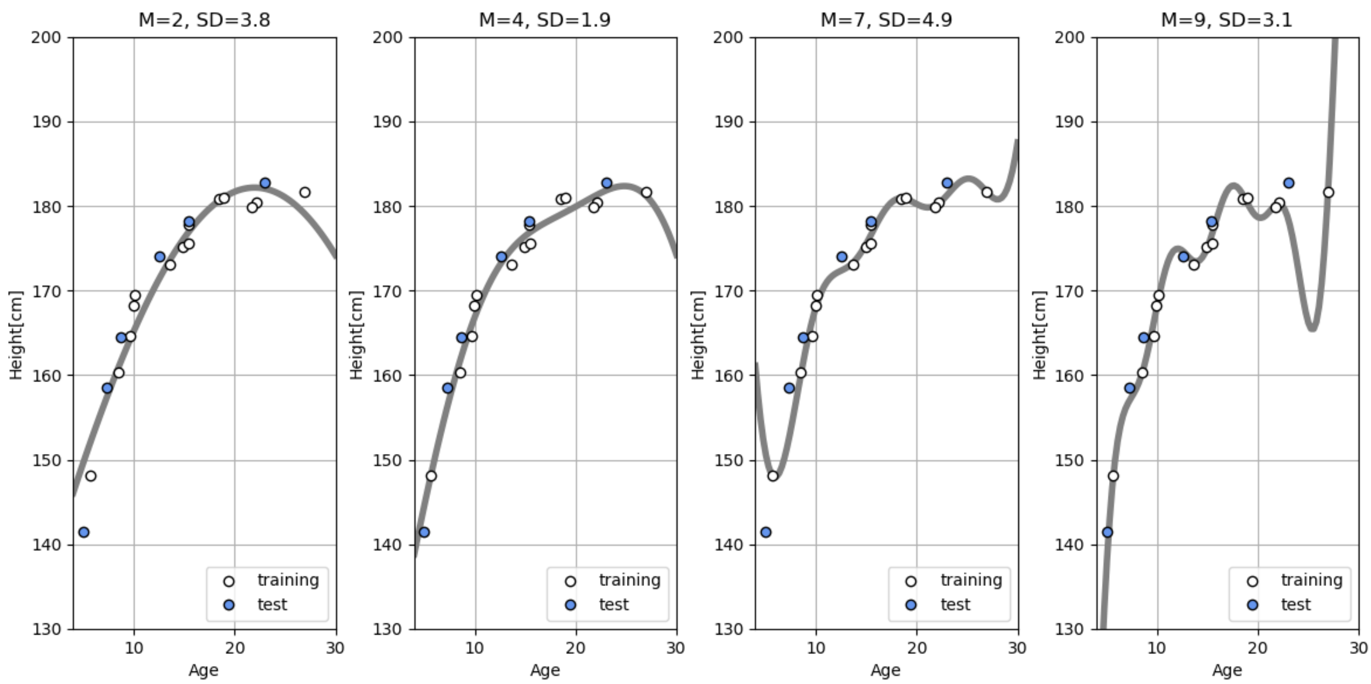
Standard deviation of error is decreasing by increasing the number of M but, the basis function is getting more curved as follow.



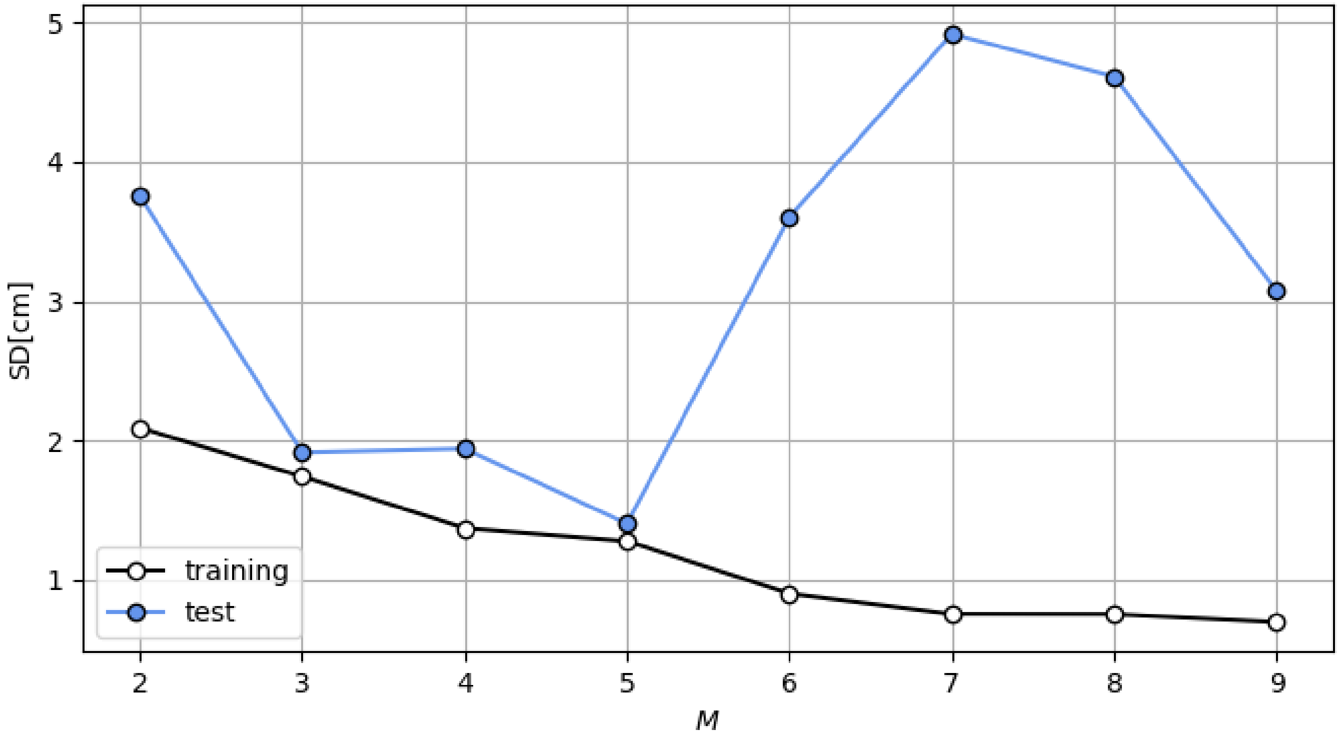
This curve gets close to each sample points but it becomes deformed at a part where there is no sample point. This is called "over-fitting". The prediction for a new data will become bad.

Hold-out validation

All of data, $\boldsymbol{x}$ and $\boldsymbol{t}$ are divided into "Test data" and "Training data". For this example, 1/4 of data is used for test and the rest, 3/4 is used for training. The parameter of model, $\boldsymbol{w}$ is optimized with only training data and a mean squared error for test data is calculated with the optimized parameter $\boldsymbol{w}$. This graph is plot by executing holdout_validation_sample.py.



In the above graphs, white points are training data and blue points are test data. If the number of M is over than 4, standard deviation for test data gets worth and over-fitting occurs.



This graph is plot by executing holdout_validation_m.py.

Cross-validation

The above result depends on how to select training data. This dependency is revealed prominently when the number of data is a few.

K-hold cross-validation

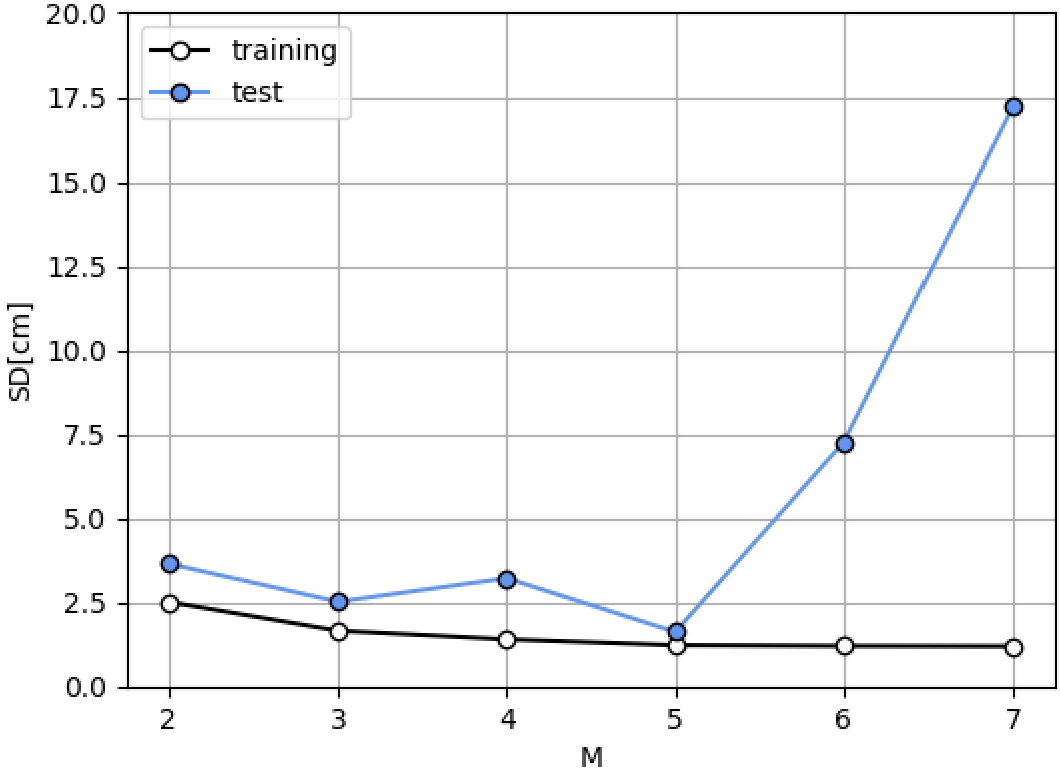
Data $\boldsymbol{X}$ and $\boldsymbol{t}$ are divided into K groups. One of them is used for test and the rest is used for training. Calculating parameters of model and mean squared error is executed for K times, and then an average of mean squared error for K times is calculated. The average value is used for validating the parameters of model.

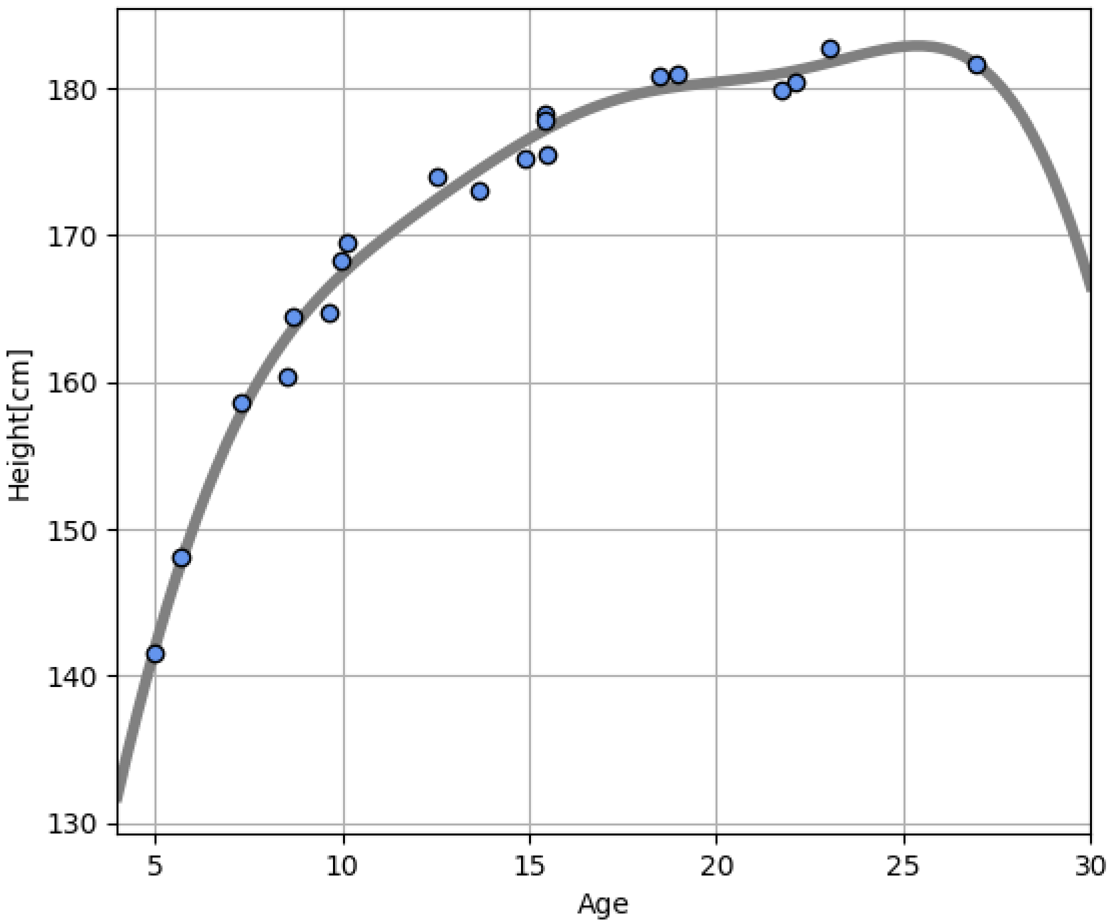
Leave-one-out cross-validation

A maximum number of division is $K = N$. In this case, a size of test data is 1. This method is called "leave-one-out cross-validation".

Validation result

This is a difference of standard deviation depending on M. When M is 5, the standard deviation is smallest. When a size of data is small, cross-validation is useful. The larger the size of data is, the longer time it takes to calculate the validation.





These graphs are plot by executing k_hold_cross_validation_m.py and learning_gaussian_function_m_5.py.

Model improvement

The above model still has a problem. It is that the graph is descending at over than 25 years old. This tendency is unusual.

Correct tendency

Height will increase gradually with age and converge at a certain age.

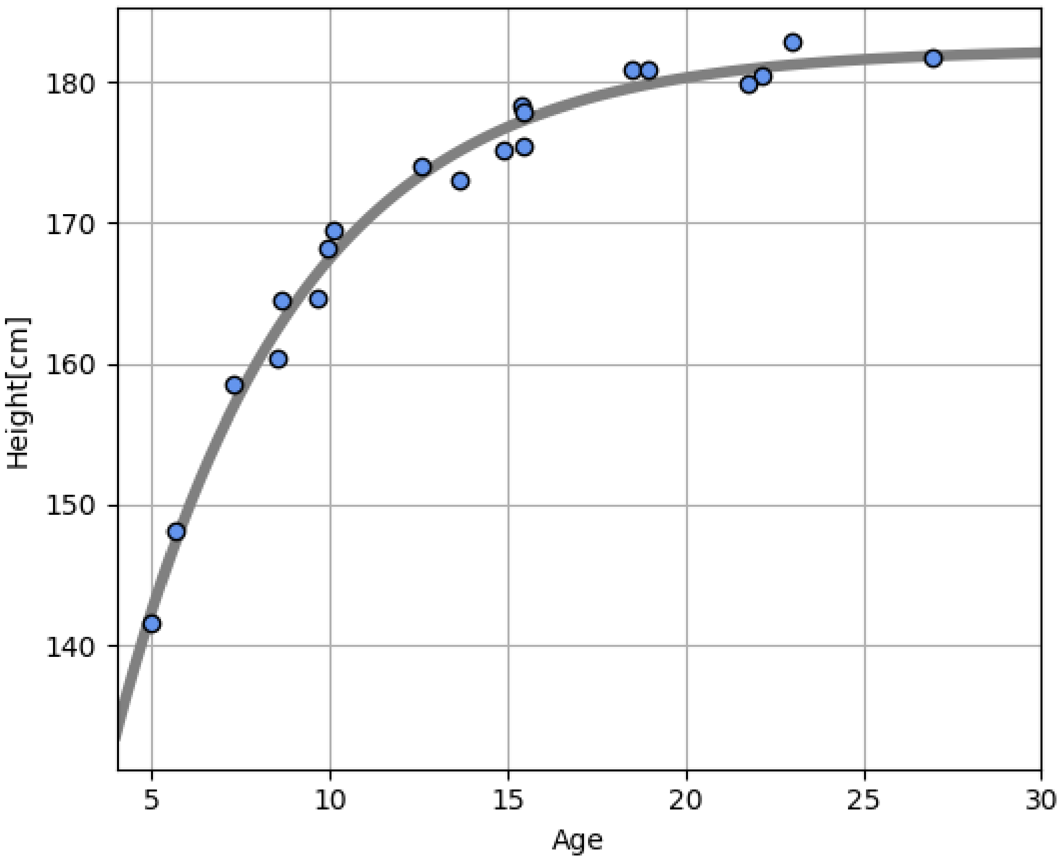
New model

$$y(x) = w_0 - w_1 exp(-w_2 x)$$

Each parameter, w_0, w_1, w_2 is a positive number. $exp(-w_2 x)$ will close to 0 when x increase. w_0 is a convergence value. w_1 is a parameter to decide a start point of the graph. w_2 is a parameter to decide a slope.

Optimization

The above parameters $\boldsymbol{w}$ is calculated by resolving optimization problem with scipy library.



These optimized parameters $\boldsymbol{w}$, $w_0 = 182.3$, $w_1 = 107.2$, $w_2 = 0.2$. Standard deviation of error is 1.31[cm]. This graph is plot by executing scipy_optimization_sample.py.

Model selection

I need to select the best model by comparing their prediction accuracy. The following model A and B are compared by leave-one-out cross-validation.

Model A

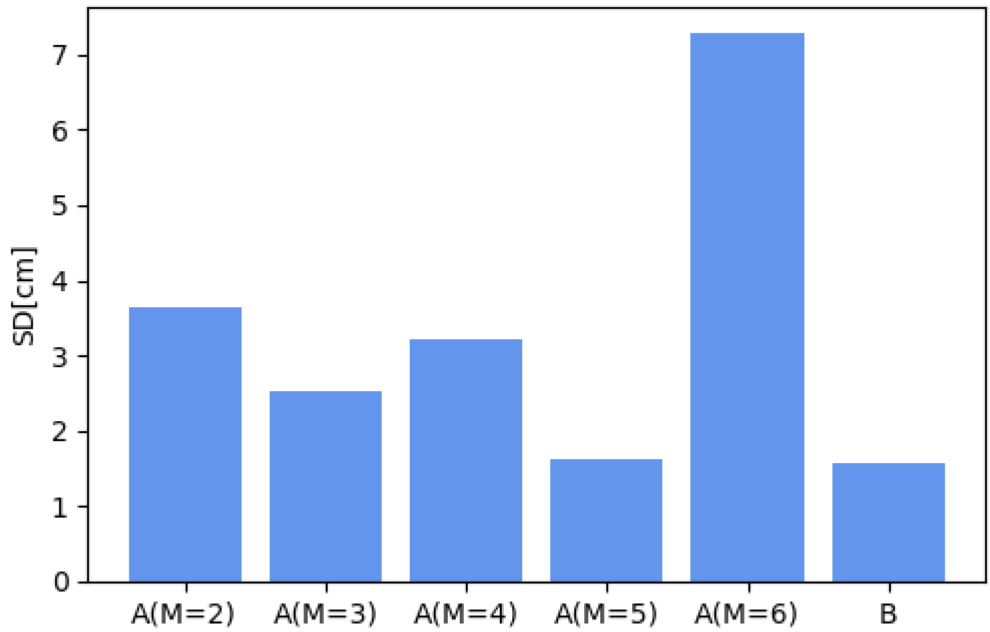
$$y(x, \boldsymbol{w}) = w_0\phi_0(x) + w_1\phi_1(x) + w_2\phi_2(x) + w_3\phi_3(x) + w_4$$
$$y(\boldsymbol{x}, \boldsymbol{w}) = \sum_{j=0}^{M} w_j\phi_j(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$$

Model B

$$y(x) = w_0 - w_1 exp(-w_2 x)$$

Comparison result

This graph is plot by executing model_comparison_cross_validation.py.



- Standard deviation(Model A): 1.63[cm]
- Standard deviation(Model B): 1.55[cm]
  According to this validation, I can conclude that Model B is more suitable to the data than Model A.

Conclusion

This is a flow of data analysis(model selection) by supervised learning.

1. We have data: input valuables and target valuables.
2. Purpose function is decided. This function is used for judging a prediction accuracy.
3. Candidates of model are decided.
4. If we choose hold out validation as a validation method, we need to devide all of data into training data and test data.
5. A parameter of each model is decided with training data in minimizing or maximizing the purpose function.
6. We predict target data from input data by each model with decided parameters and the model which the error is the smallest.