Regression

Liang Liang

Regression

- Regression is a subcategory of supervised learning where the goal is to predict continuous target value given the features of a sample.

- The relationship between the features and the target value may be linear or nonlinear.

- The target value could be a scalar or a vector.

Notation

- a training set of N data points $\{x_1, x_2, x_3, \dots, x_N\}$ and $x_n \in \mathcal{R}^M$
- a data point $x_n \in \mathcal{R}^M$, it is a vector and has M elements
- a set of 'ground-truth' target values $\{y_1, y_2, y_3, \dots, y_N\}$

- Each data point has M features

$$x_n = \left[x_{(n,1)}, x_{(n,2)}, x_{(n,3)}, \dots, x_{(n,m)}, \dots, x_{(n,M)}\right]^T$$

- Drop the index n

$$x = \left[x_{(1)}, x_{(2)}, x_{(3)}, \dots, x_{(m)}, \dots, x_{(M)}\right]^T$$

x_1 is a data point/vector

$x_{(1)}$ is a feature component of a vector, it is a scalar

Linear Regression

$\hat{y} = a_0 + a_1 x_{(1)} + a_2 x_{(2)} + a_3 x_{(3)}$, a linear model
$y = \hat{y} + \varepsilon$

a data point x is a feature vector $\left[x_{(1)}, x_{(2)}, x_{(3)}\right]$

y is the 'true' target value (ground-truth)

$\hat{y}$ is the predicted target value from the model

ε is something that can not be explained by the linear model

ε is treated as 'random noise'

$\{a_0, a_1, a_2, a_3\}$ are the parameters of the linear model

Linear Regression

simple linear regression: $\hat{y} = a_0 + a_1 x_{(1)}$

a_0 is intercept

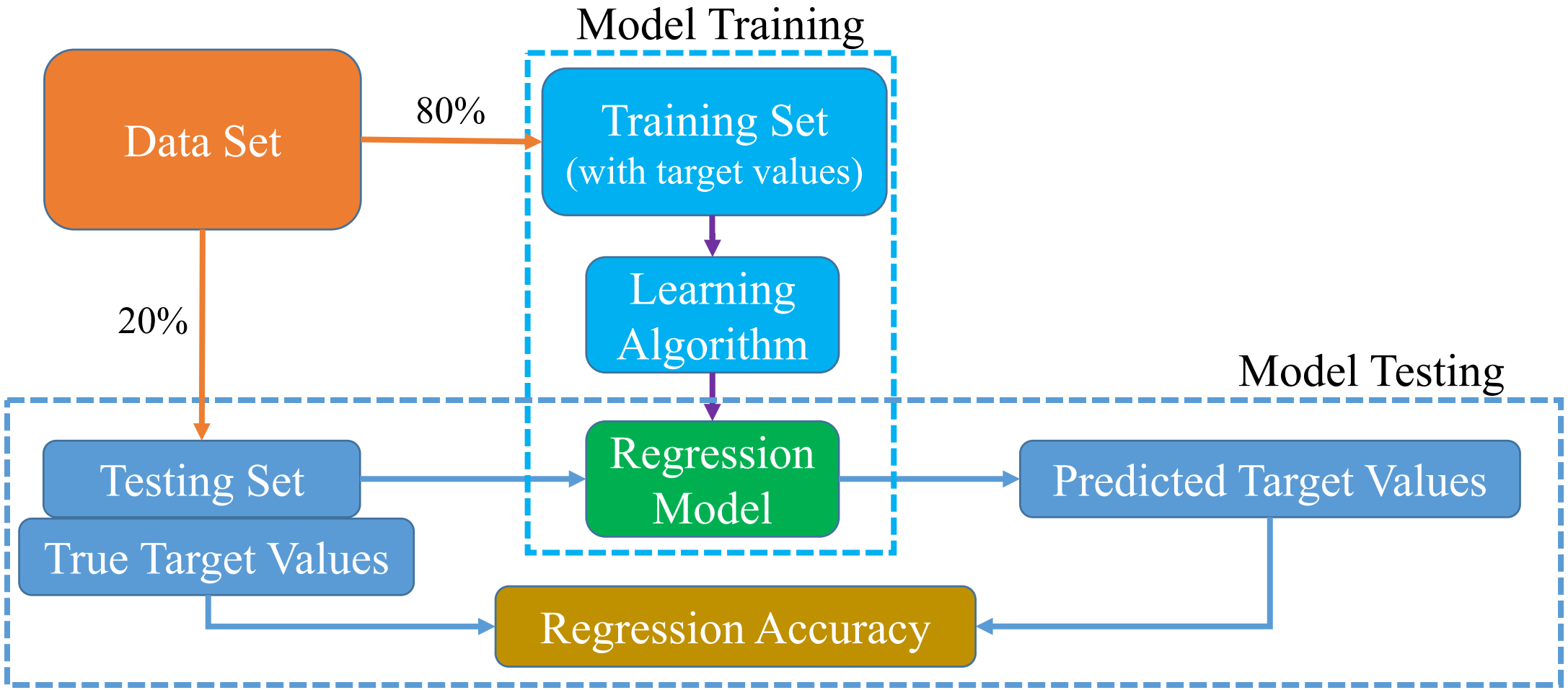
a_1 is slope

multiple linear regression: $\hat{y} = a_0 + a_1 x_{(1)} + a_2 x_{(2)} \ldots + a_M x_{(M)}$

Fit the linear model to training dataset, to obtain the optimal parameters

Evaluate the trained/fitted linear model on testing dataset

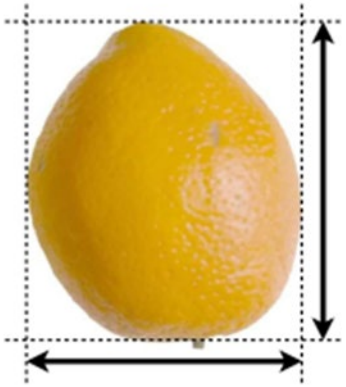
the workflow of a regression study



Example: linear regression on the fruit dataset

A bucket of fruits

The fruit dataset was created by Dr. Iain Murray at the University of Edinburgh. He bought a few dozen oranges, lemons and apples, and recorded their features in a table.



The fruit dataset {1:apple, 2:mandarin, 3:orange, 4:lemon},
Each row contains the information of a fruit sample/instance

fruit label	fruit_name	subtype	mass (g)	width (cm)	height (cm)	color_score
1	apple	granny_smith	192	8.4	7.3	0.55
4	lemon	spanish_belsan	194	7.2	10.3	0.70

http://usapple.org/the-industry/apple-varieties/

Example: linear regression on the fruit dataset

first step: load the dataset

The table has 59 rows (samples)

```
fruits = pd.read_table('fruit_data_with_colors.txt')
```

```
fruits
```

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
3	2	mandarin	mandarin	86	6.2	4.7	0.80
4	2	mandarin	mandarin	84	6.0	4.6	0.79
5	2	mandarin	mandarin	80	5.8	4.3	0.77
6	2	mandarin	mandarin	80	5.9	4.3	0.81
7	2	mandarin	mandarin	76	5.8	4.0	0.81
8	1	apple	braeburn	178	7.1	7.8	0.92
9	1	apple	braeburn	172	7.4	7.0	0.89

linear regression on the fruit dataset
goal: predict mass given width, height and color

fruit label	fruit_name	subtype	mass (g)	width (cm)	height (cm)	color_score
1	apple	granny_smith	192	8.4	7.3	0.55

Feature Vector x_n

$x_{(n,1)}$: width
$x_{(n,2)}$: height
$x_{(n,3)}$: color

Linear Model

Target

mass $\hat{y}_n$

n is the index of the sample
(row index in the table)

$$\hat{y}_n = a_0 + a_1 x_{(n,1)} + a_2 x_{(n,2)} + a_3 x_{(n,3)}$$

```
# instance of the regressor
linear_model = LinearRegression(fit_intercept=True)
```

linear regression on the fruit dataset - data splitting

split the data (59 samples) into a training dataset (80%) and a testing dataset (20%)

```
feature_names = ['width', 'height', 'color_score']
feature_names
```

```
['width', 'height', 'color_score']
```

```
target_name = ['mass']
target_name
```

```
['mass']
```

Split the data into a Training dataset and a Testing dataset

```
X = fruits[feature_names]
Y = fruits[target_name]
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
```

Model Training: fit the model to the training dataset

The training set contains N input-output pairs: $\{(x_n, y_n), n = 1, \dots, N\}$

x_n is a feature vector ; y_n is the true target value

MSE (mean squared error) loss function:

$$L(a_0, a_1, a_2, a_3) = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2$$

$$\hat{y}_n = a_0 + a_1 x_{(n,1)} + a_2 x_{(n,2)} + a_3 x_{(n,3)}$$

The goal of training is to find the best parameters $\{a_0, a_1, a_2, a_3\}$ such that the loss function is minimized. After training, the prediction $\hat{y}_n$ from the model should be very close to the true target y_n

Model Testing: apply the model to the testing dataset

The testing set contains K input-output pairs: $\{(x_k, y_k), k = 1, \dots, K\}$

mean squared error (MSE)

$$MSE = \frac{1}{K}\sum_{k=1}^{K}(y_k - \hat{y}_k)^2$$

mean absolute error (MAE)

$$MAE = \frac{1}{K}\sum_{k=1}^{K}|y_k - \hat{y}_k|$$

Mean absolute percentage error (MAPE)

$$MAPE = \frac{1}{K}\sum_{k=1}^{K}\left|\frac{y_k - \hat{y}_k}{y_k}\right| \times 100\%$$

```
#prediction on the testing dataset
Y_test_pred = linear_model.predict(X_test)
MSE = np.mean((Y_test - Y_test_pred)**2)
MAE = np.mean(np.abs(Y_test - Y_test_pred))
MAPE =  np.mean(np.abs(Y_test - Y_test_pred)/Y_test)
print('MSE=', MSE)
print('MAE=', MAE)
print('MAPE=', MAPE)
```

we do not need for loops, use vectorized operation

```
MSE= 638.9054197256988
MAE= 18.544529450261397
MAPE= 0.10798195018512798
```

Model Testing: apply the model to the testing dataset

The testing set contains K input-output pairs: $\{(x_k, y_k), k = 1, \dots, K\}$

coefficient of determination R^2

$$R^2 = 1 - \frac{\sum_{k=1}^{K}(y_k - \hat{y}_k)^2}{\sum_{k=1}^{K}(y_k - \bar{y})^2}, \text{ where } \bar{y} = \frac{1}{K}\sum_{k=1}^{K} y_k$$

```
#coefficient of determination R^2 on training set
linear_model.score(X_train, Y_train)
```

```
0.8523395191909264
```

```
#coefficient of determination R^2 on testing set
linear_model.score(X_test, Y_test)
```

```
0.8466217984120755
```

The MSE loss function ~ least mean square (LMS)

- Linear regression using MSE loss is also called least mean square fitting
- Given N <u>training</u> data points <u>with</u> target values $\{(x_n, y_n), n = 1, \dots, N\}$, find the best parameters that minimize the MSE loss:

$$L(a_0, \dots, a_M) = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2$$

$$\hat{y}_n = a_0 + a_1 x_{(n,1)} + a_2 x_{(n,2)} + \cdots + a_m x_{(n,m)} + \cdots + a_M x_{(n,M)}$$

- To obtain the optimal a_i, as usual, we compute $\frac{\partial L}{\partial a_i}$, and set it to 0

The MSE loss function ~ least mean square (LSM)

$$L(a_0, \dots, a_K) = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2$$

$$\hat{y}_n = a_0 + a_1 x_{(n,1)} + a_2 x_{(n,2)} + \cdots + a_m x_{(n,m)} + \cdots + a_M x_{(n,M)}$$

- we compute $\frac{\partial L}{\partial a_i}$ and set it to 0

$$\frac{\partial L}{\partial a_i} = -\frac{2}{N}\sum_{n=1}^{N}\left(y_n - \sum_{m=0}^{M} a_m x_{(n,m)}\right)x_{(n,i)} = 0$$

Then we obtain:

$$\sum_{n=1}^{N}\sum_{m=0}^{M} a_m x_{(n,m)} x_{(n,i)} = \sum_{n=1}^{N} y_n x_{(n,i)}$$

But:

It is not obvious to get an equation like $a_i = f(\{x_n, y_n, \hat{y}_n\})$,

let's try vector form of the derivative

- find the best parameters that minimize the MSE loss:

$$L = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2$$

$$\hat{y}_n = a_0 + a_1 x_{(n,1)} + a_2 x_{(n,2)} + \cdots + a_m x_{(n,m)} + \cdots + a_M x_{(n,M)}$$

$$x_n = \left[1, x_{(n,1)}, x_{(n,2)}, x_{(n,3)}, \ldots, x_{(n,m)}, \ldots, x_{(n,M)}\right]^T$$

$$\hat{y}_n = w^T x_n$$

$$w = [a_0, a_1, a_2, \ldots, a_M]^T$$

compute $\frac{\partial L}{\partial w}$, and set it to 0

$$\boldsymbol{L(w) = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2 \text{ and } \hat{y}_n = w^T x_n}$$

- compute $\frac{\partial L}{\partial w}$, and set it to 0

$$\frac{\partial L}{\partial w} = -\frac{2}{N}\sum_{n=1}^{N}\left(y_n - w^T x_n\right)x_n = 0$$

- we obtain: $\sum_{n=1}^{N} x_n (x_n)^T w = \sum_{n=1}^{N} y_n x_n$

Define: $X = [x_1, x_2, \dots, x_N]$, $Y = [y_1, y_2, \dots, y_N]^T$

Then: $XX^T = \sum_{n=1}^{N} x_n (x_n)^T$, and $XY = \sum_{n=1}^{N} y_n x_n$

Therefore: $XX^T w = XY \quad => \quad w = \left(XX^T\right)^{-1} XY$

If $\left(XX^T\right)^{-1}$ does not exit, we can pseudo inverse of XX^T

- Given N <u>training</u> data points <u>with</u> target values $\{(x_n, y_n), n = 1, \dots, N\}$, find the model parameters that minimize the MSE loss:

$$L(w) = \frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n)^2$$

$$\hat{y}_n = w^T x_n$$
$$w = [a_0, a_1, a_2, \dots, a_M]^T$$
$$x_n = \left[1, x_{(n,1)}, x_{(n,2)}, x_{(n,3)}, \dots, x_{(n,m)}, \dots, x_{(n,M)}\right]^T$$

- **We could use gradient descent algorithms to find the optimal parameter w**

- We could use gradient descent to find the optimal parameter w

  step-0: initialize w randomly

  step-1: compute $\frac{\partial L}{\partial w} = -\frac{2}{N}\sum_{n=1}^{N}\left(y_n - w^T x_n\right)x_n$ which is a function of w

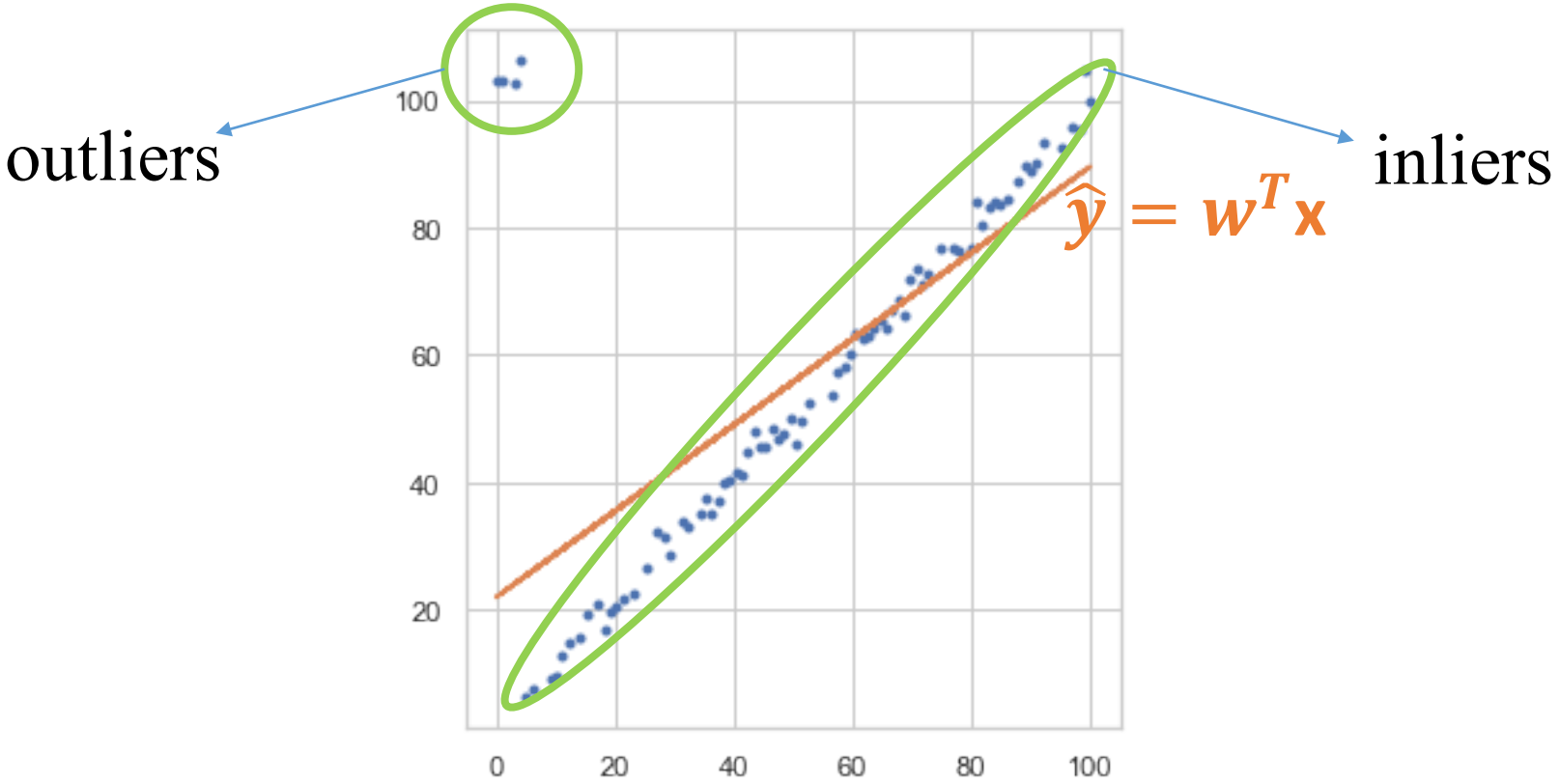
  step-2: update $w \leftarrow w - \eta\frac{\partial L}{\partial w}$ where η is learning rate

  repeat step-1 and step-2 until the algorithm converges

Robust Linear Regression

- $\hat{y}_n = w^T x_n,$ $\quad$ x_n is a data point
- $y_n = \hat{y}_n + \varepsilon_n,$ $\quad$ ε_n is the noise assuming i.i.d. from a Gaussian $\mathcal{N}(0, \sigma^2)$
- What happens if some errors $\{\varepsilon_n\}$ are very large ?

  and the assumption of i.i.d. Gaussian $\mathcal{N}(0, \sigma^2)$ is not valid ?



Robust Linear Regression

- $\hat{y}_n = w^T x_n,$ $\quad$ x_n is a data point
- $y_n = \hat{y}_n + \varepsilon_n,$ $\quad$ ε_n is the noise
- What happens if some errors $\{\varepsilon_n\}$ are very large ?

  and the assumption of i.i.d. Gaussian $\mathcal{N}(0, \sigma^2)$ is not valid ?

- Two approaches:

(1) modify the loss function to make it robust (less sensitive) to outliers

e.g. HuberRegressor

(2) classify data points as outliers and inliers, and fit the model only to inliers

e.g. RANdom SAmple Consensus (RANSAC)

Robust Linear Regression - Huber Regressor

- The loss function of Huber Regressor

$$L(w,\sigma) = \sum_{n=1}^{N}\left(\sigma + h\left(\frac{y_n - w^T x_n}{\sigma}\right)\sigma\right) + \alpha\|w\|_2^2$$

where

$$h(z) = \begin{cases} z^2 & , if\ |z| < \epsilon \\ 2\epsilon|z| - \epsilon^2 & , otherwise \end{cases}$$

$z = \frac{y_n - w^T x_n}{\sigma}$, *normalized error,* $\epsilon = 1.35$ (default value in sk-learn)

idea: improve robustness by reducing the sensitivity of the loss with respect to the data samples that are outliers:

if (x_n, y_n) is an outlier, then the error z is very large

but $h(z)$ is relatively small

Robust Linear Regression - Huber Regressor

- The loss function of Huber Regressor

$$L(w,\sigma) = \sum_{n=1}^{N}\left(\sigma + h\left(\frac{y_n - w^T x_n}{\sigma}\right)\sigma\right) + \alpha\|w\|_2^2$$

The MSE loss function

$$L(w) = \frac{1}{N}\sum_{n=1}^{N}\left(\frac{y_n - w^T x_n}{\sigma}\right)^2$$

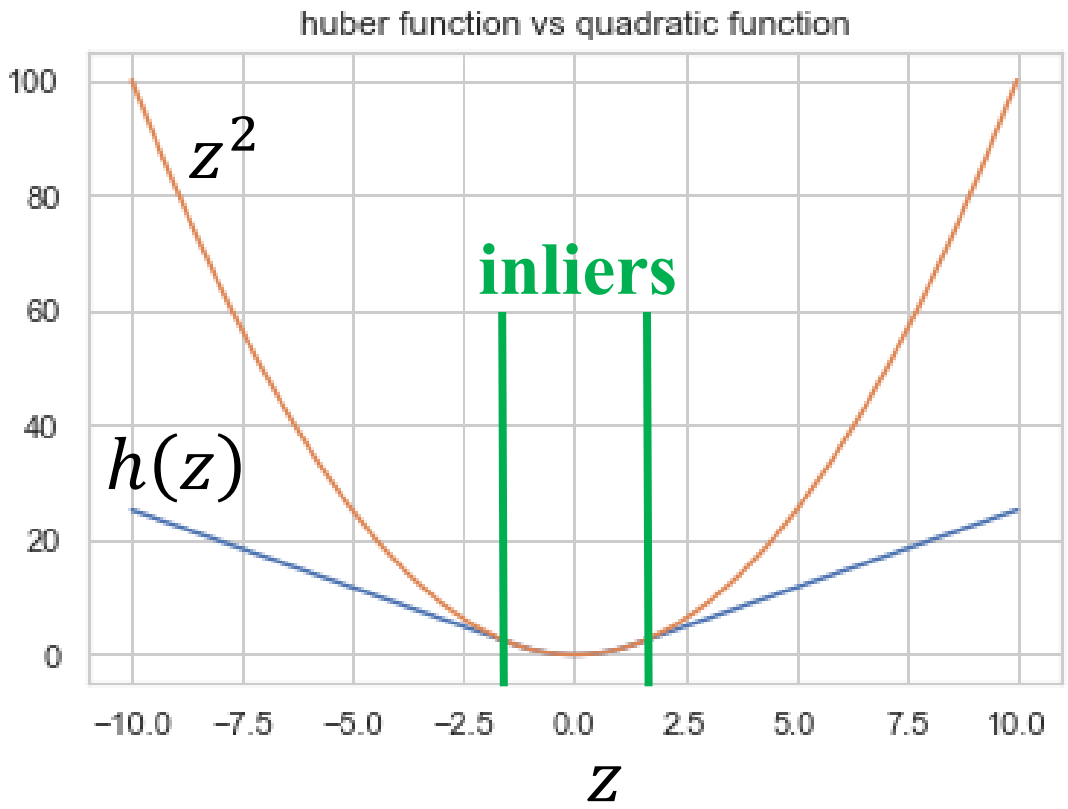
where

$$h(z) = \begin{cases} z^2 & , if\ |z| < \epsilon \\ 2\epsilon|z| - \epsilon^2 & , otherwise \end{cases}$$

$$z = \frac{y_n - w^T x_n}{\sigma}, \ normalized\ error$$

$if\ (x_n, y_n)$ is an outlier, then z is very large

but $h(z)$ is relatively small



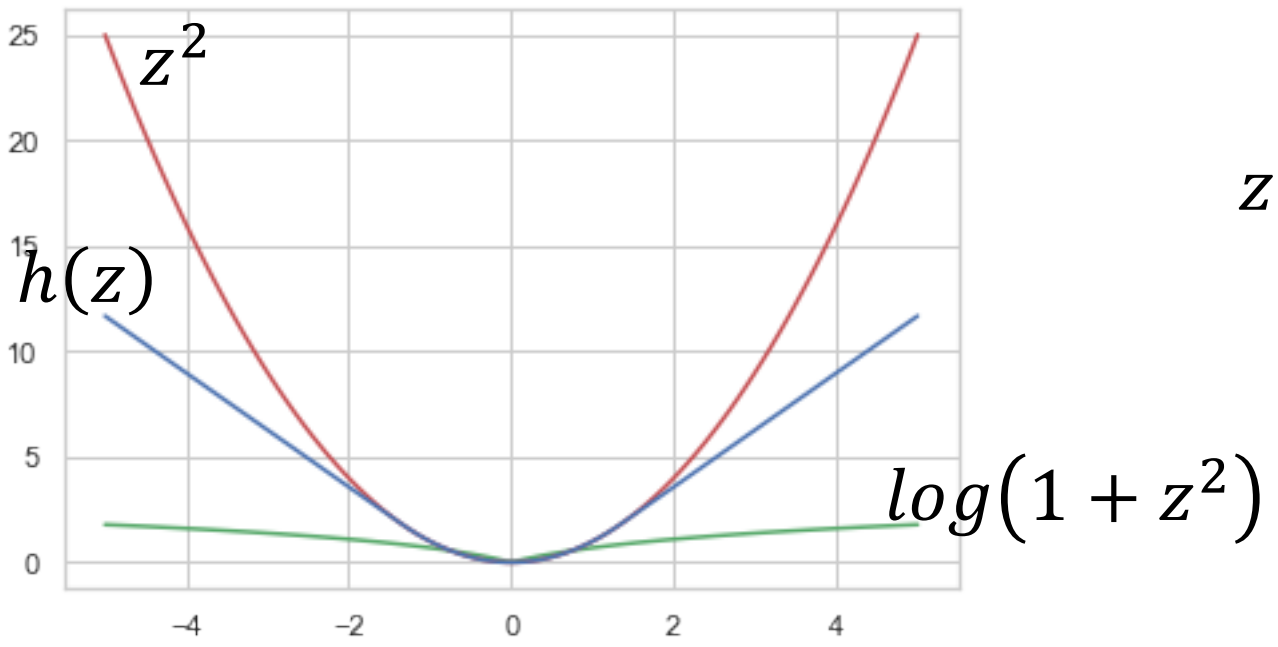
Robust Linear Regression - Huber Regressor

- The loss function of Huber Regressor

$$L(w,\sigma) = \sum_{n=1}^{N}\left(\sigma + h\left(\frac{y_n - w^T x_n}{\sigma}\right)\sigma\right) + \alpha\|w\|_2^2$$

Robust Linear Regression - log loss

$$L(w,\sigma) = \frac{1}{N}\sum_{n=1}^{N} log\left(1 + \left(\frac{y_n - w^T x_n}{\sigma}\right)^2\right)$$



$$z = \frac{y_n - w^T x_n}{\sigma}, \textit{ normalized error}$$

You can implement this loss by yourself

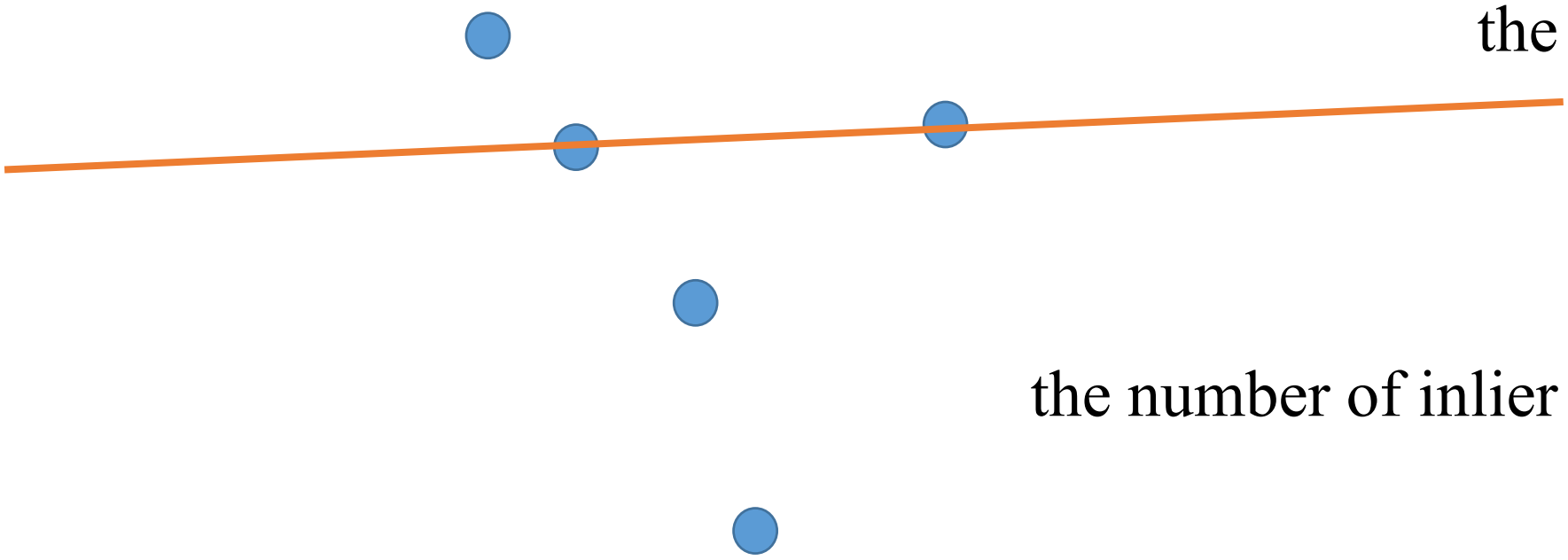
Robust Linear Regression - RANSAC

- Idea: classify data points as outliers and inliers, and fit the model to inliers
- It needs many iterations, and four steps in one iteration:
  - (1) Select some data points <u>randomly</u> from the original data points
  - (2) Fit a model to selected data points
  - (3) Classify all data points as inliers or outliers using the fitted model
  - (4) Count the number of inlier data points
- The best model has the maximum number of inlier data points
- Classification of inliers and outliers is based on absolute residual

  if $|y_n - \hat{y}_n| \leq threshold$, then data point (x_n, y_n) is inlier

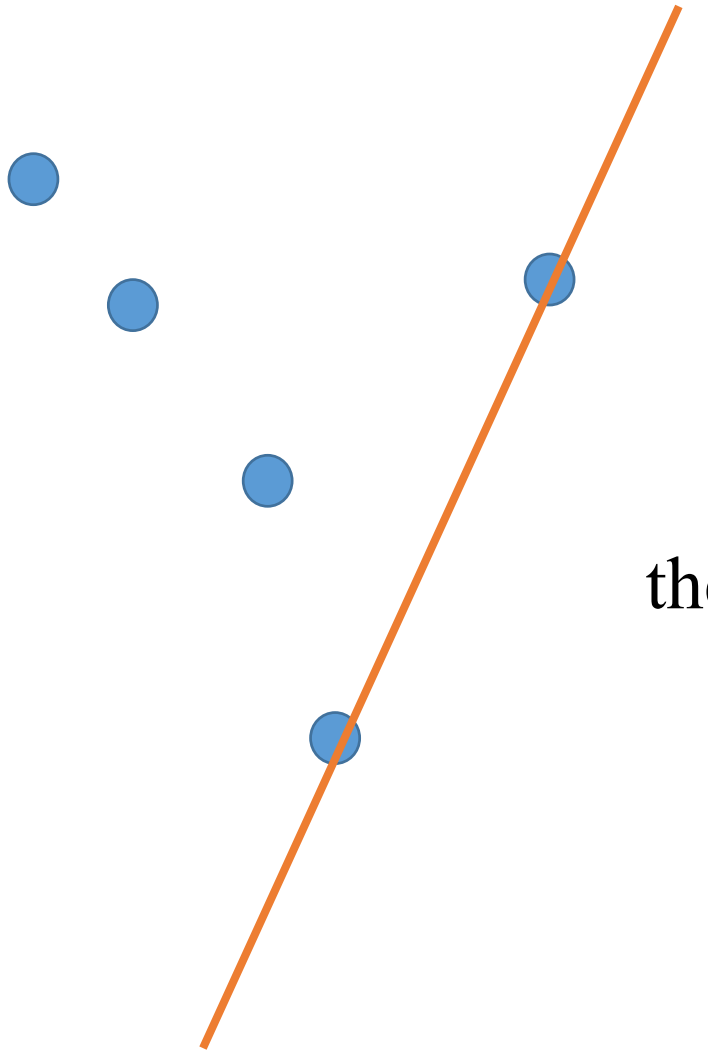
  $threshold$ could be the median absolute deviation of the target values

Robust Linear Regression - RANSAC



the number of inlier data points = 2

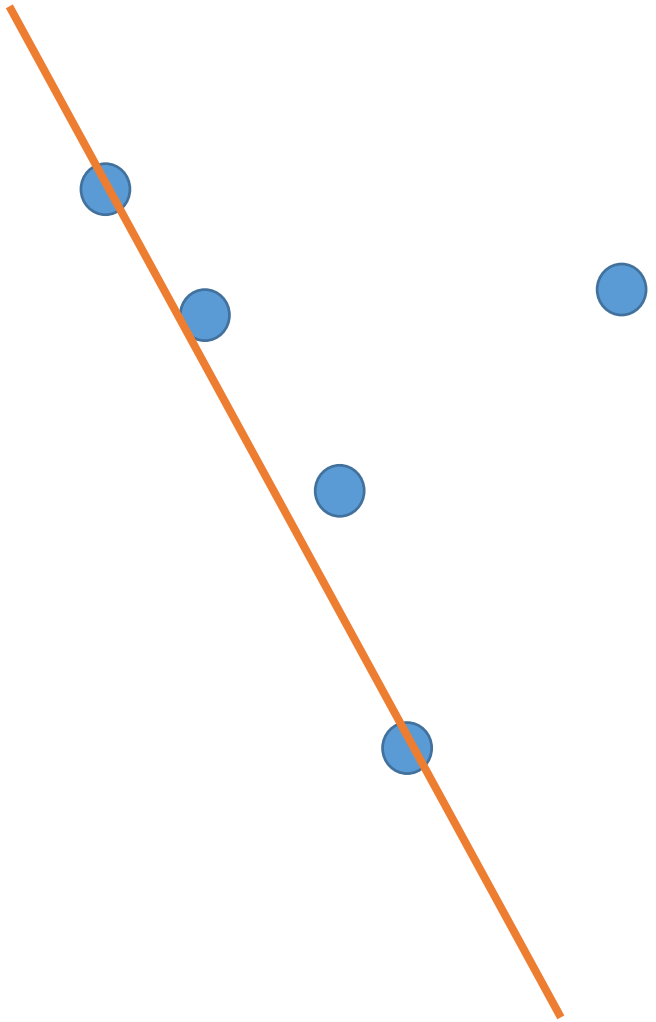
Robust Linear Regression - RANSAC



the second model

the number of inlier data points = 2

Robust Linear Regression - RANSAC



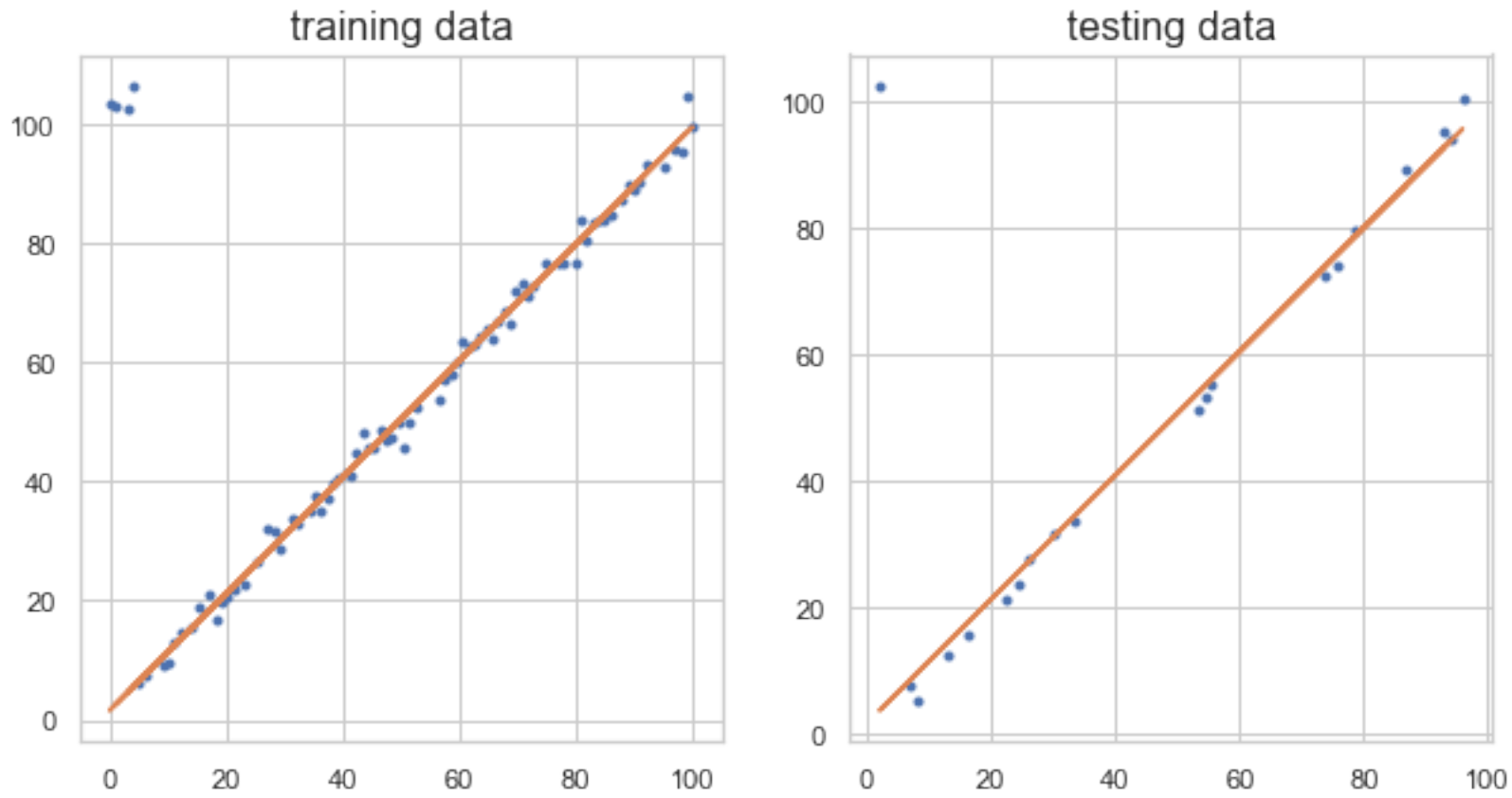
the third model

the number of inlier data points = 4

the third model is the best if the max number of iterations is set to 3

Robust Linear Regression - RANSAC

- RANSAC can also be used for nonlinear regression.
- It is a non-deterministic algorithm producing only a reasonable result with a certain probability given the maximum number of iterations.



Polynomial Regression (1D input, 1D output)

- Polynomial basis functions $f_k(x) = x^k$
- x is a data point in 1D space, $x \in \mathcal{R}$

$$\hat{y} = a_0 + a_1 x + a_2 x^2 + \cdots + a_{100} x^{100}$$

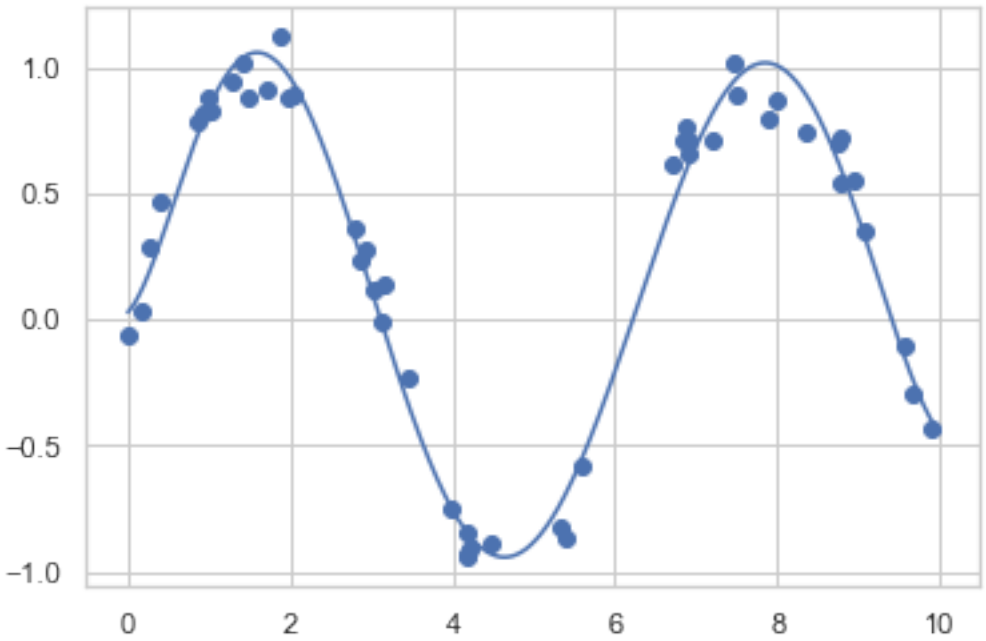
- Convert the polynomial model to a linear model:

$$\hat{y} = a_0 + a_1 \tilde{x}_{(1)} + a_2 \tilde{x}_{(2)} + \cdots + a_{100} \tilde{x}_{(100)}$$

$\tilde{x} = \left[\tilde{x}_{(1)}, \tilde{x}_{(2)}, \ldots, \tilde{x}_{(100)}\right]^T = \left[x, x^2, \ldots, x^{100}\right]^T$ is a data point in $\mathcal{R}^{100}$

Polynomial Regression

fit a polynomial of degree 7 to data points generated from a sine function



data points are generated from

$$y = \sin(x) + \varepsilon$$

$$\varepsilon \sim \mathcal{N}(0, 0.1)$$

Nonlinear_Regression_using_Basis_Functions.ipynb

Nonlinear Regression using Gaussian Basis Functions (high-dimensional input, 1D output)

- Gaussian basis functions $f_k(x) = \mathcal{N}(x|\mu_k, \Sigma_k)$
- x could be a data point in a high dimensional space (e.g. $x \in \mathcal{R}^{100}$)
- The nonlinear regression model is

$$\hat{y} = a_1\mathcal{N}(x|\mu_1, \Sigma_1) + \cdots + a_{100}\mathcal{N}(x|\mu_{100}, \Sigma_{100})$$

- Convert the above nonlinear model into a linear model

$$\hat{y} = a_1\tilde{x}_{(1)} + a_2\tilde{x}_{(2)} + \cdots + a_{100}\tilde{x}_{(100)}$$

$$\tilde{x} = \left[\tilde{x}_{(1)}, \tilde{x}_{(2)}, \ldots, \tilde{x}_{(100)}\right]^T$$

$$= [\mathcal{N}(x|\mu_1, \Sigma_1), \mathcal{N}(x|\mu_2, \Sigma_2), \ldots, \mathcal{N}(x|\mu_{100}, \Sigma_{100})]^T$$

It is a data point in $\mathcal{R}^{100}$

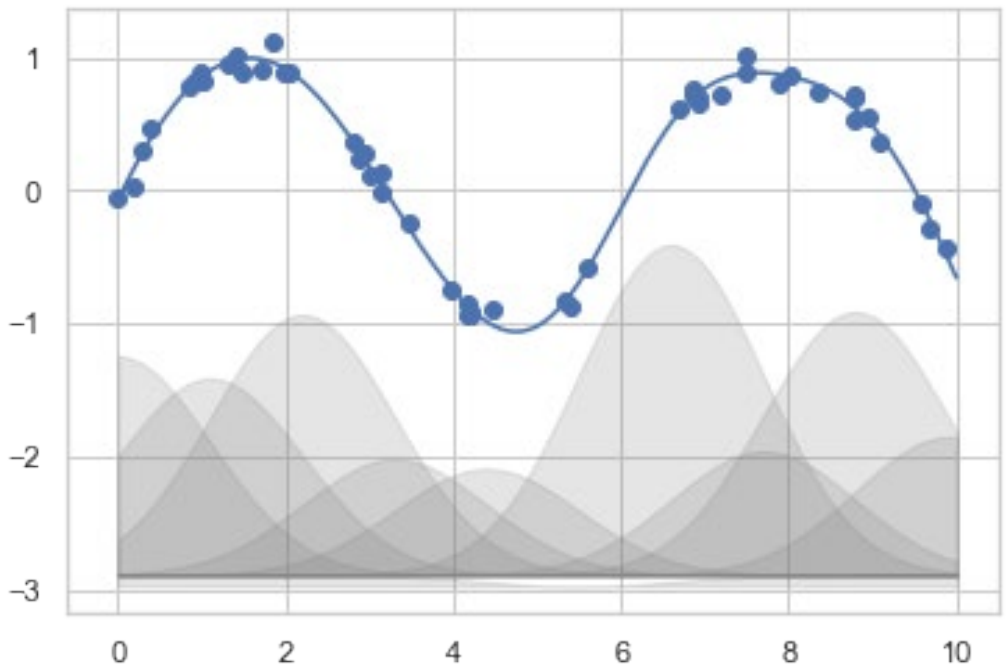
Regression using 1D Gaussian Basis Functions

Gaussian basis functions $f_k(x) = \mathcal{N}(x|\mu_k, \sigma^2)$, x refers to a 1D data point

$$\hat{y} = a_1\mathcal{N}\left(x\middle|\mu_1, \sigma^2\right) + \cdots + a_{10}\mathcal{N}\left(x\middle|\mu_{10}, \sigma^2\right)$$

μ_k, σ are predefined in this example

we only need to estimate $\{a_1, \ldots a_{10}\}$ from the data



data points are generated from

$y = \sin(x) + \varepsilon$

$\varepsilon \sim \mathcal{N}(0, 0.1)$