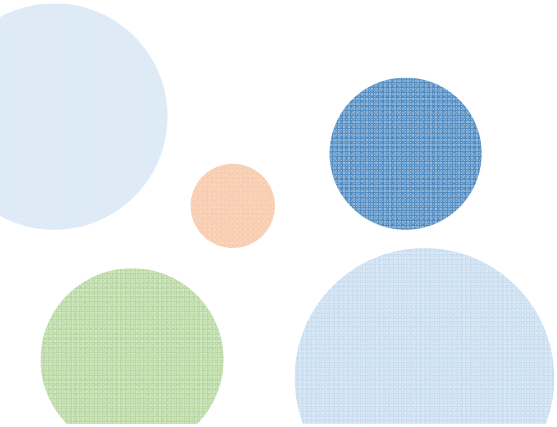
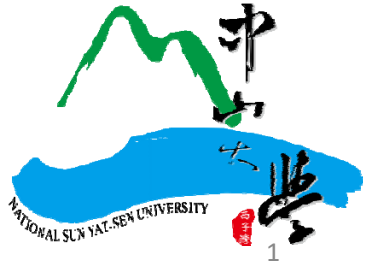
Basic Machine Learning (1)
Linear-regression

國立中山大學
資訊工程系
張雲南

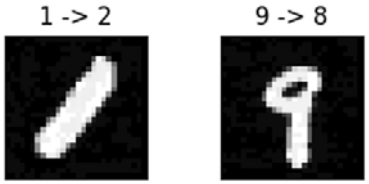




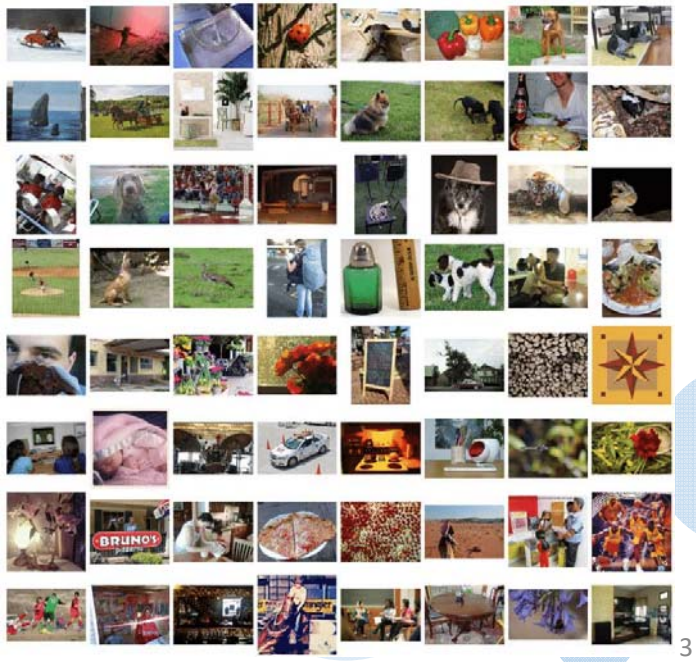
Outlines

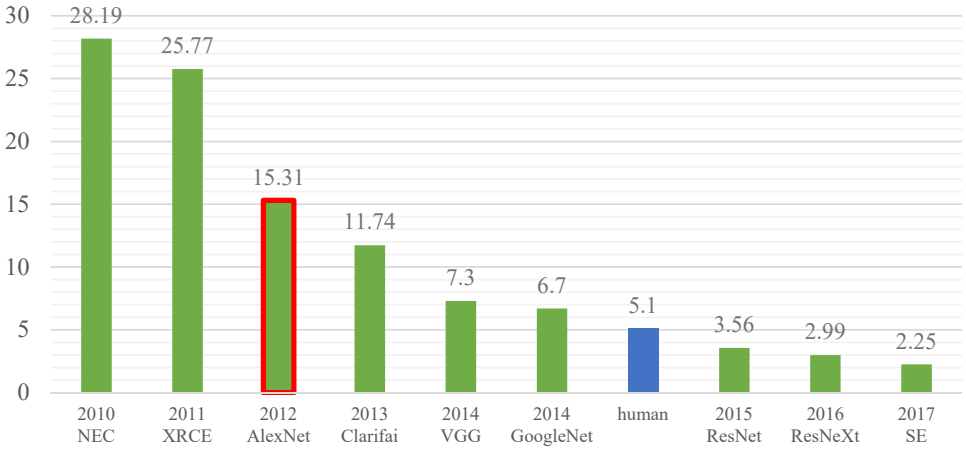
- Introduction to machines learning
- Linear Regression
- Data preprocessing
- Using scikit-learn for machine learning

Right perception of current AI tech



14 million images, 20000 categories



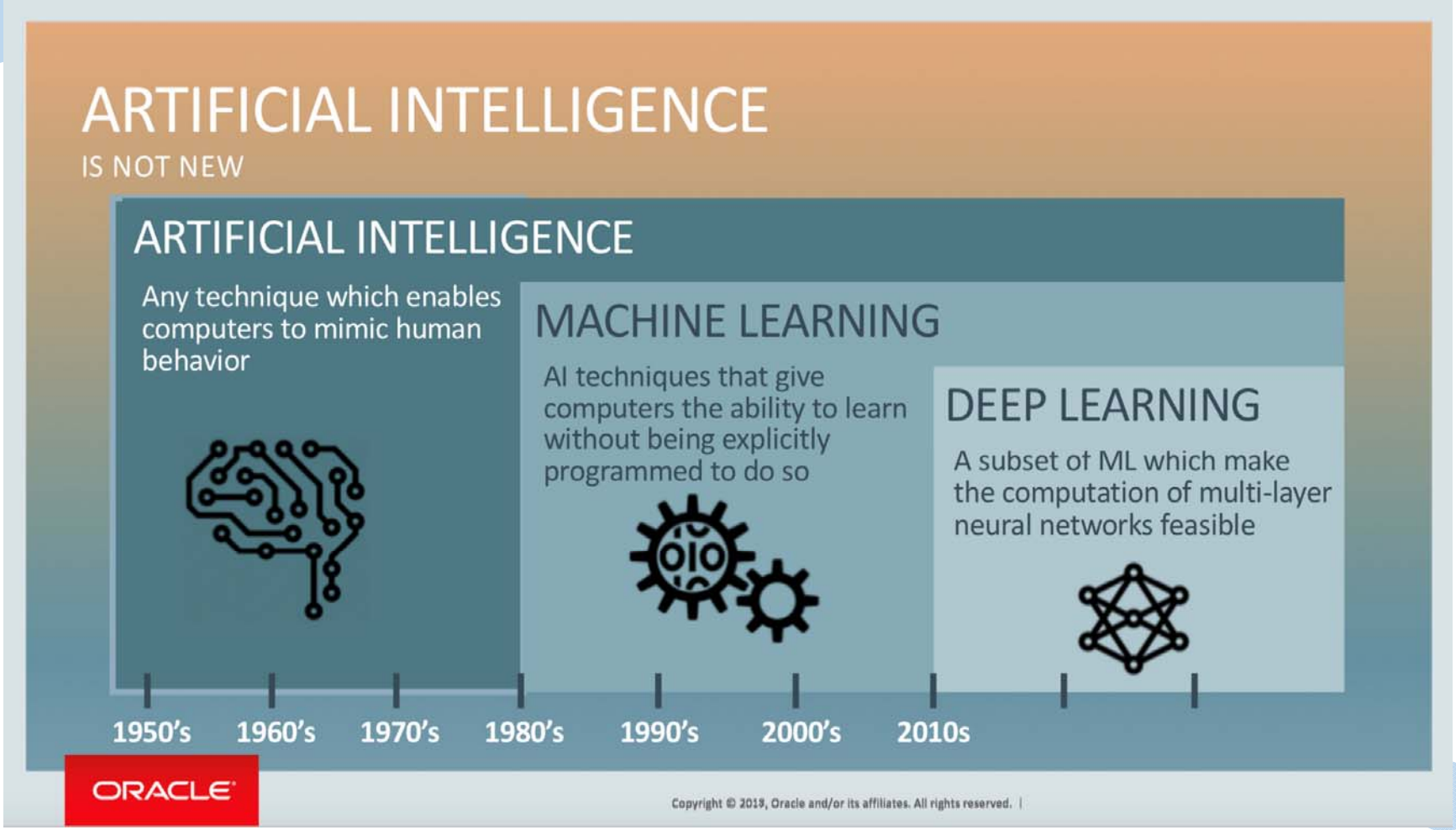


ImageNet Large Scale Visual Recognition Competition

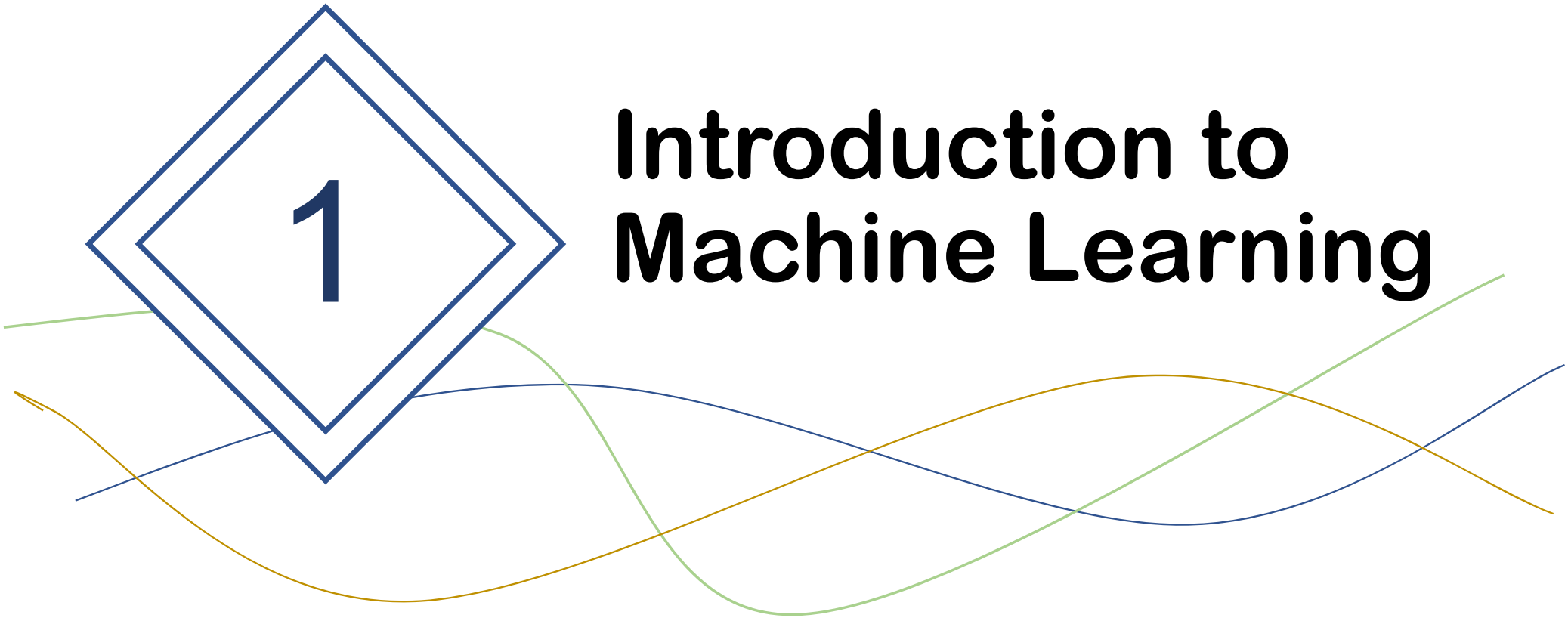
Three driving forces for AI

- 數據 Data
- 演算法 Algorithm ★
  - Machine learning model (Neural Network architecture)
  - Training schemes and methods
- 計算能力 Computing Power
  - Tesla M40 GPU: 14 days, nVidia DGX (8 x P100 GPU): 21hr
    - Training Data : ImageNet 1K, ~1M image, 1000 class
    - ResNet-50, 90 epochs, Top-1 Accuracy : 72%

AI vs ML vs DL



圖源: https://blogs.oracle.com/bigdata/difference-ai-machine-learning-deep-learning



Introduction to Machine Learning

Machine Learning

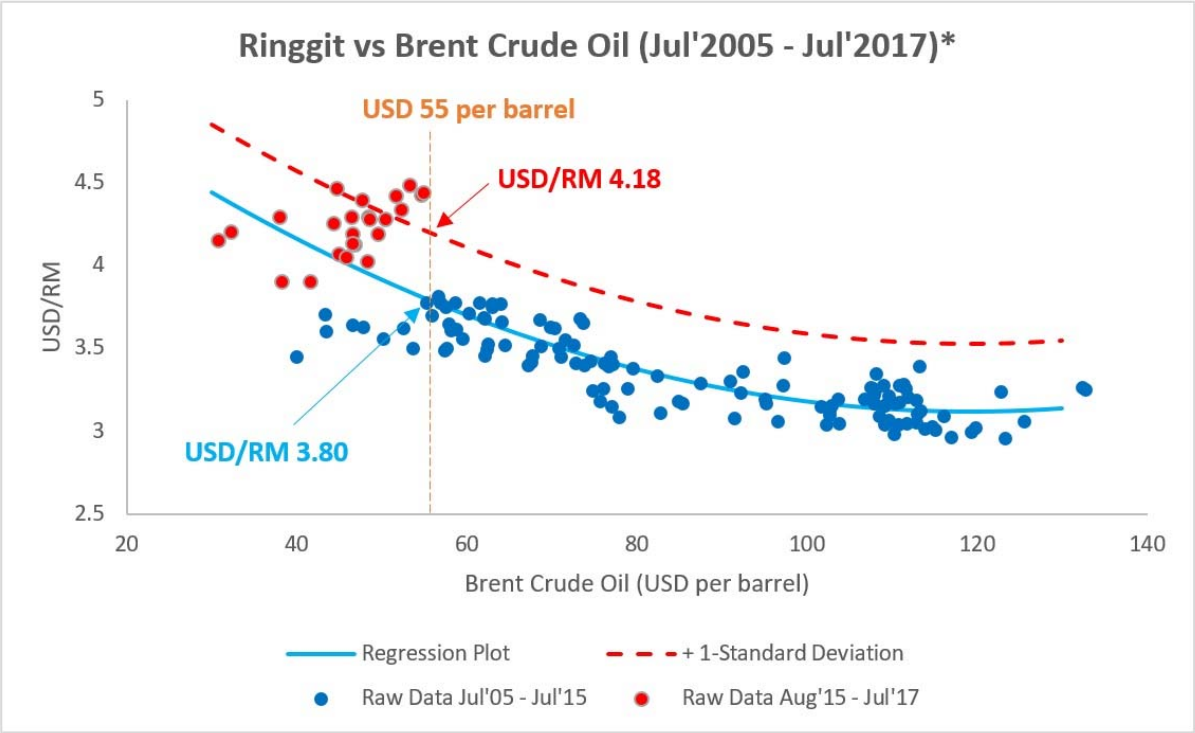
- ***Train*** a ***model*** by ***data*** to learn the mapping of ***features (X)*** to the ***results (Y)***.

$$f: X \rightarrow Y$$

  - The ***model*** is used to realize the mapping function, and will be implemented by the software.
  - Depending on the relationship between ***X*** and ***Y***, different types of ***models*** may be chosen.
    - f: 身高 → 體重 => *can choose linear regression model*
    - f: 客戶資料 → 信用核可 => *decision tree model may be better*
  - The dimension of ***X*** and ***Y*** may be low or high, but they all have to be encoded by numbers.

Regression

- $y = f(x_1, x_2, \ldots, x_n)$
- Target y is a float.
- For example:
  - Oil price forecast
  - House price estimation

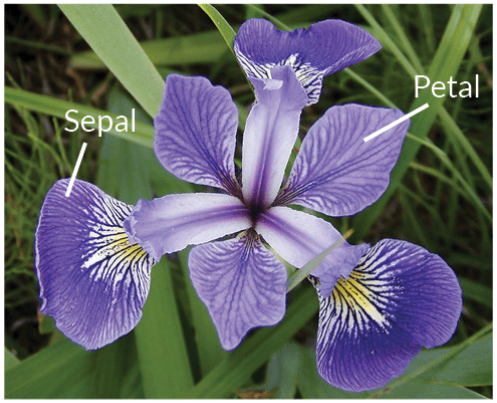


* *The data range was chosen from July 2005 onwards because the Ringgit was unpegged after July 2005.*

Use *data* to learn *hypothesis functions* based on different machine-learning *models*.

Classification

- $y = f(x_1, x_2, \ldots, x_n)$
- Target y is a binary vector or discrete number, which corresponds to some categories.
- For example:
  - Iris flowers classification
  - Yes/NO, Live/Dead, Positive/negative,…..



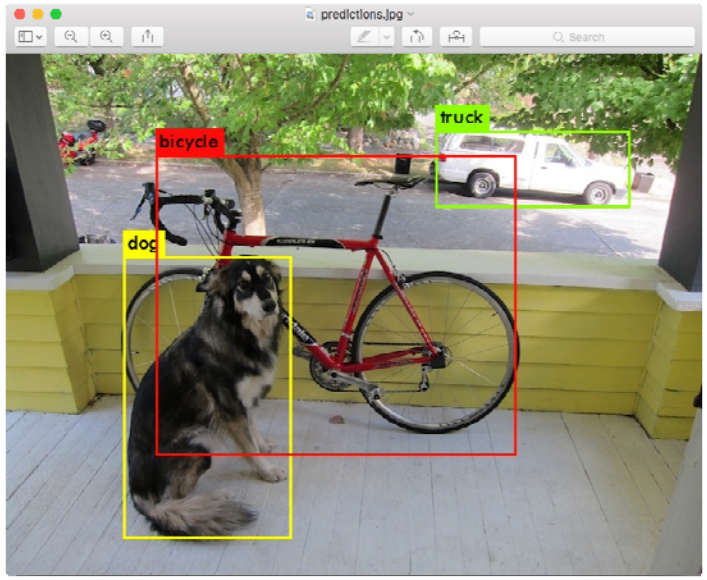
Iris Versicolor

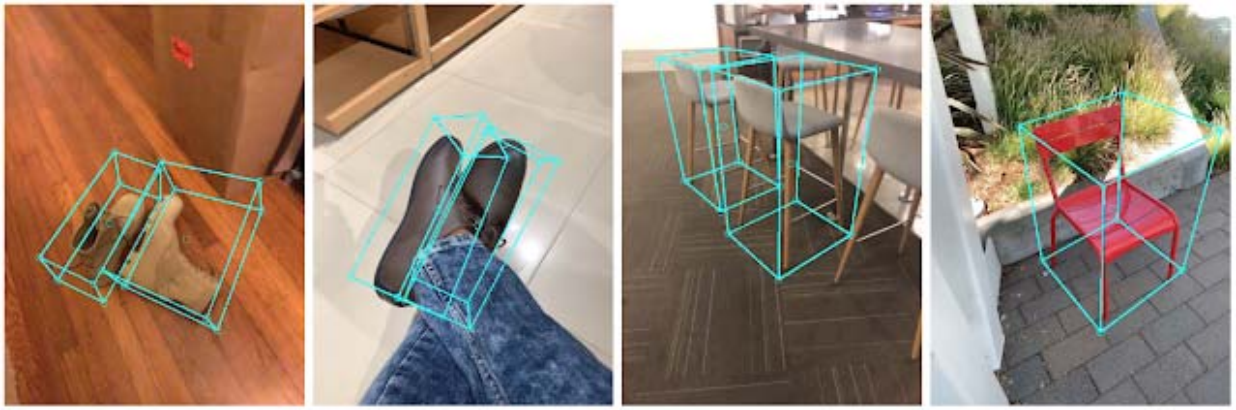

Iris Setosa


Iris Virginica

Pattern recognition

- $y = f(x_1, x_2, \ldots, x_n)$
- Target y is a list of triplets ***(class, w, h)***
- For example:
  - Object Detection





Data generation

- $y = f(x_1, x_2, \ldots, x_n)$
- Target y is high-dimensional data.

(1.2,0.8) → G → 

Hello → G → I'm fine

 → G → 

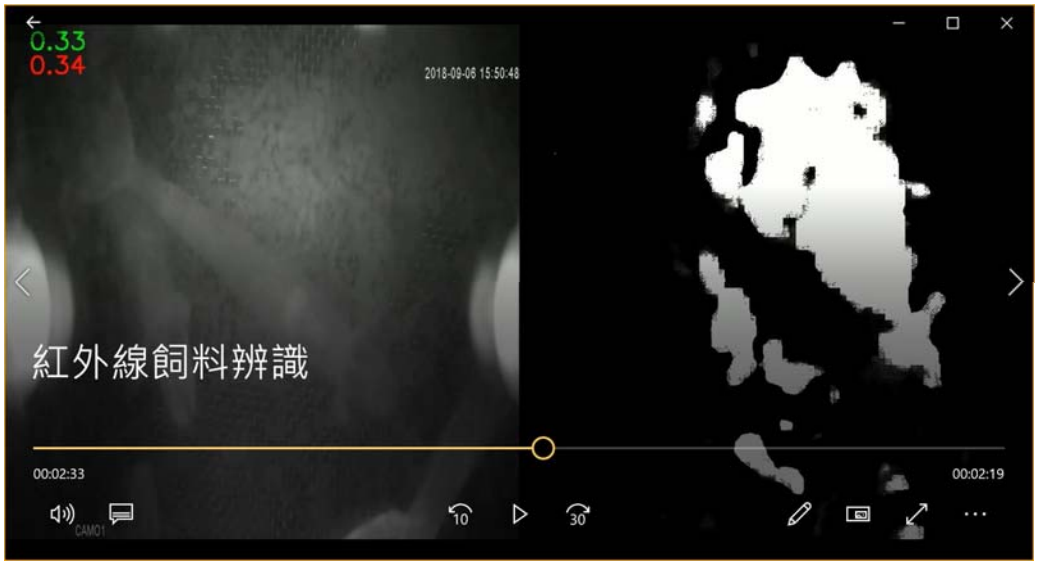
→ G → 

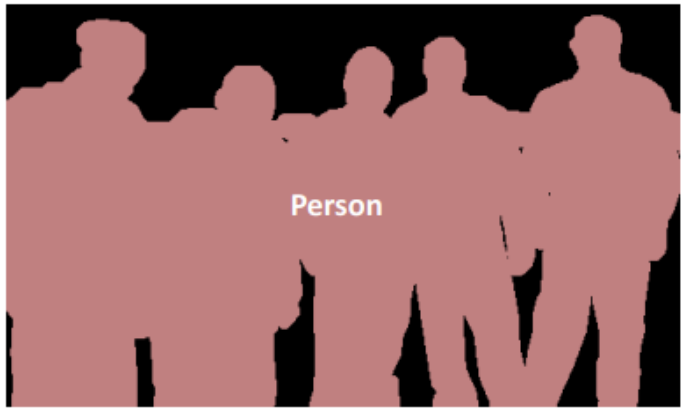
this bird is red with white and has a very short beak → G → 

 → G → 

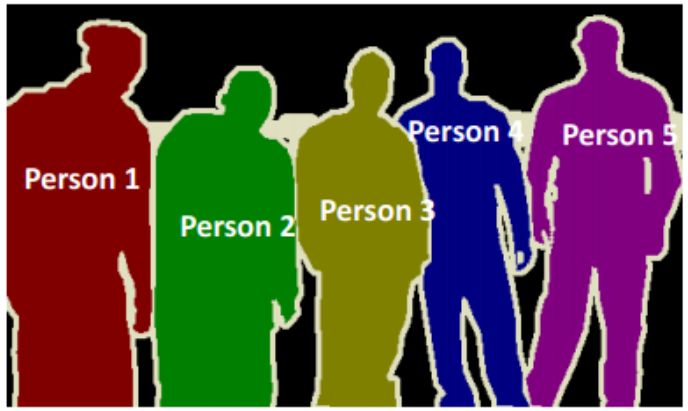
Segmentation

- $y = f(x_1, x_2, \ldots, x_n)$
- Target y is a high-dimensional vector consisting of discrete numbers.





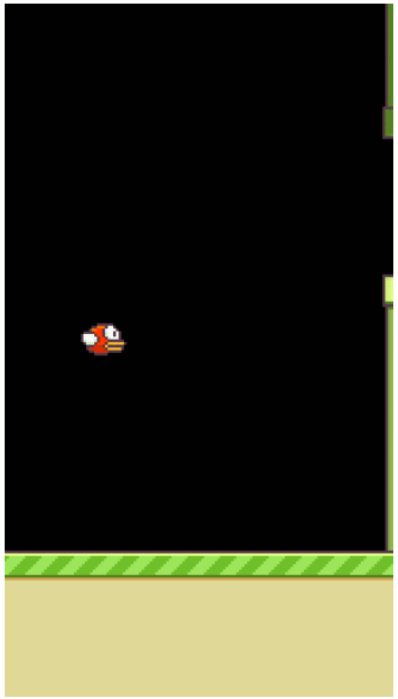
Semantic Segmentation



Instance Segmentation

Action choice

- $y = f(x_1, x_2, \ldots, x_n)$
- Target y corresponds to the actions to be taken.
- For example:
  - AlphaGo
  - Play video game



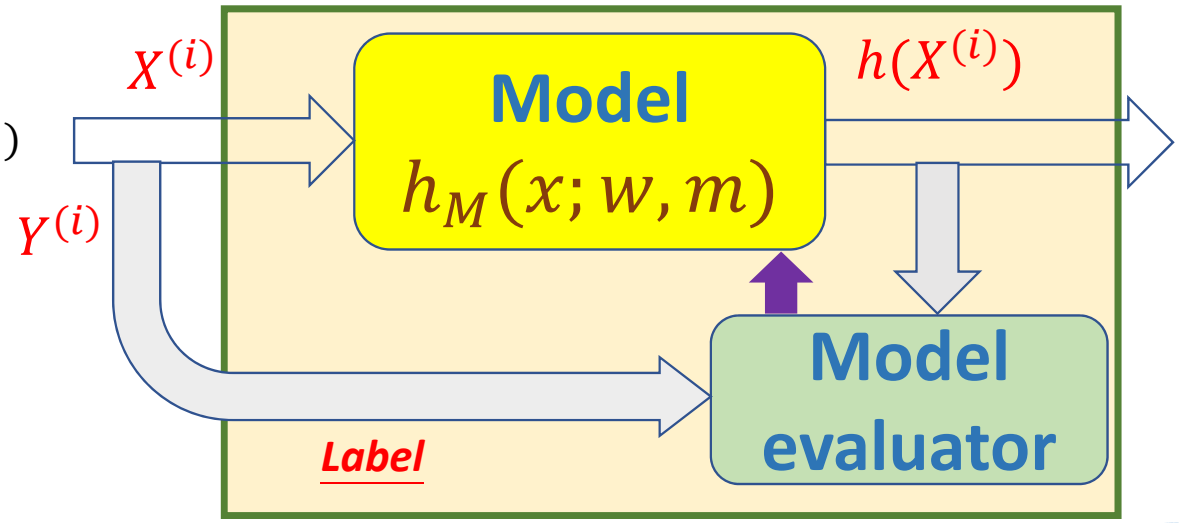
Supervised learning

Training stage

dataset

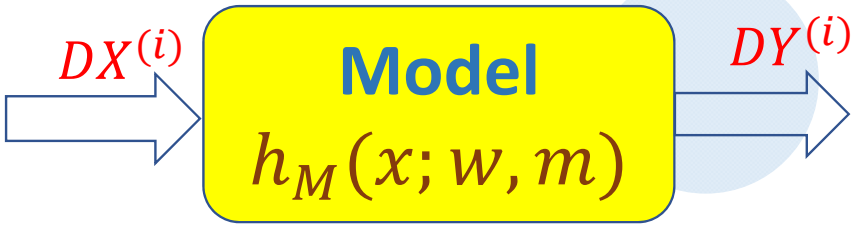
$$f: X \rightarrow Y$$

$$(X^{(N)}, Y^{(N)}) \cdots\cdots (X^{(2)}, Y^{(2)}) \ (X^{(1)}, Y^{(1)})$$



** Depend on whether the object function includes "label".*

Inference stage



Supervised learning

Training stage

X_2

Y_2 : Monkey

X_1

Y_1 : Monkey

X_0

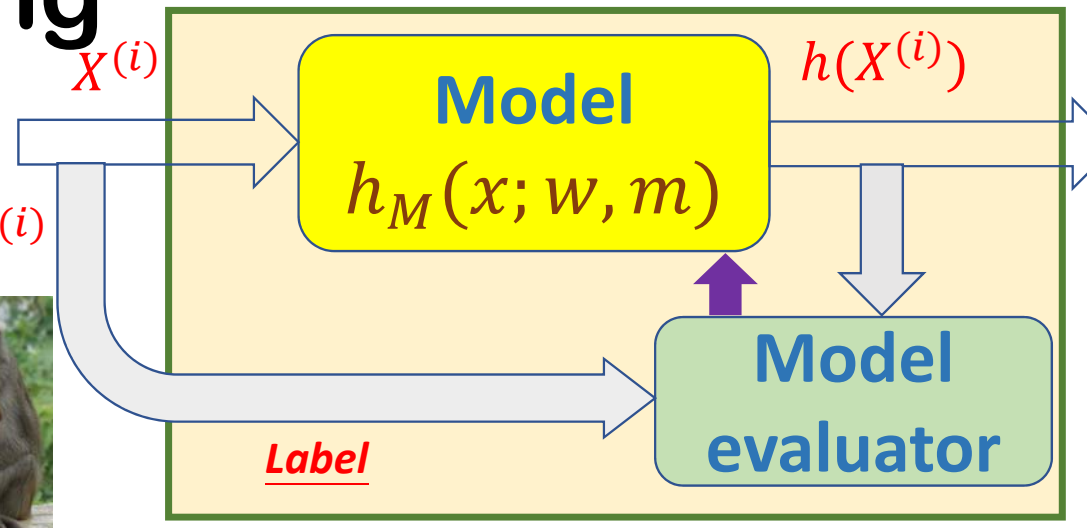
Y_0 : Monkey

X_4

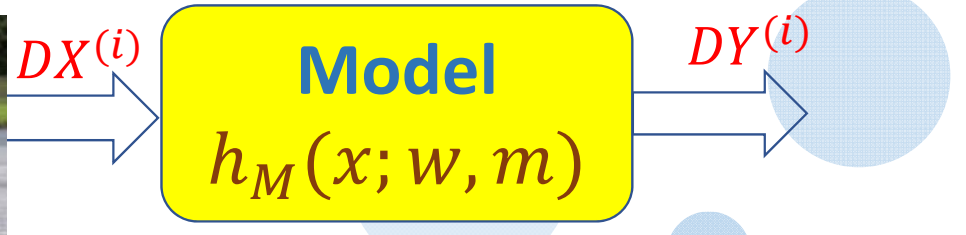
Y_4 : Dog

X_3

Y_3 : Dog

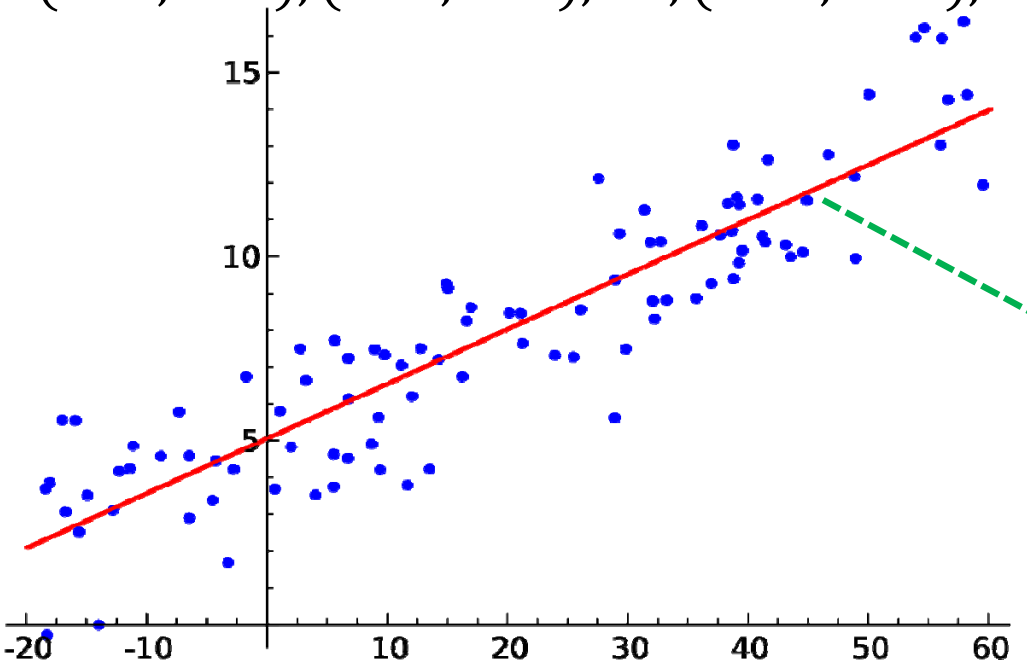


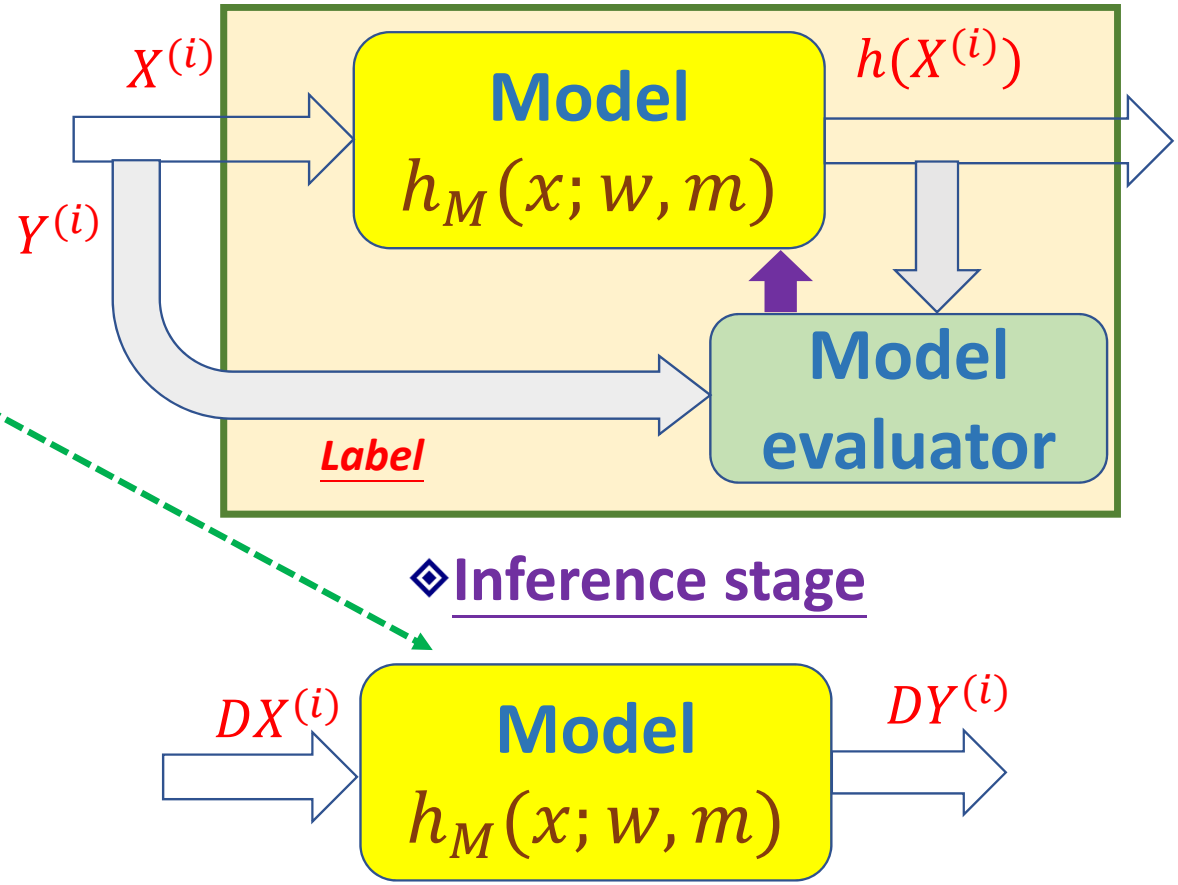
Inference stage



Supervised learning for regression

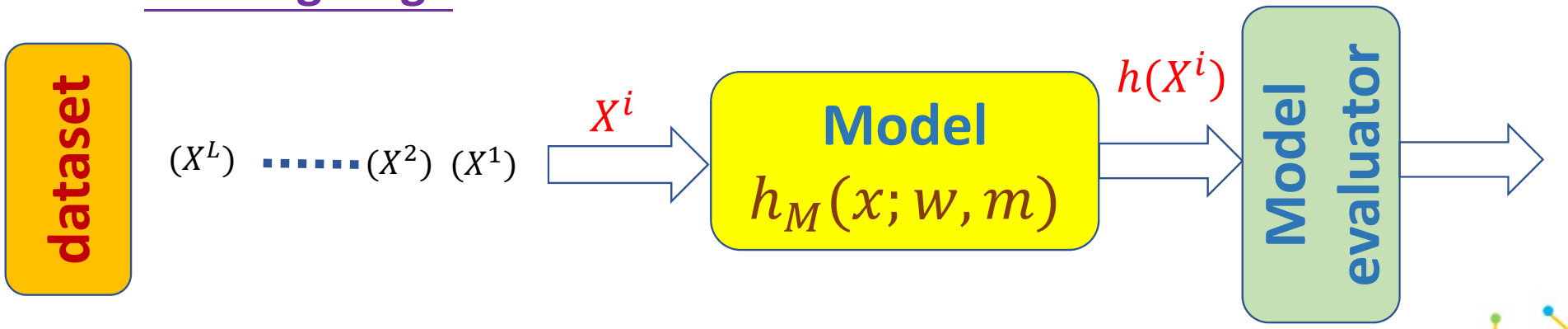
$(X^{(0)}, Y^{(0)}), (X^{(1)}, Y^{(1)}), \ldots, (X^{(N)}, Y^{(N)}),$



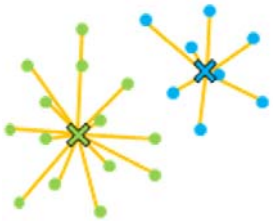


Unsupervised learning

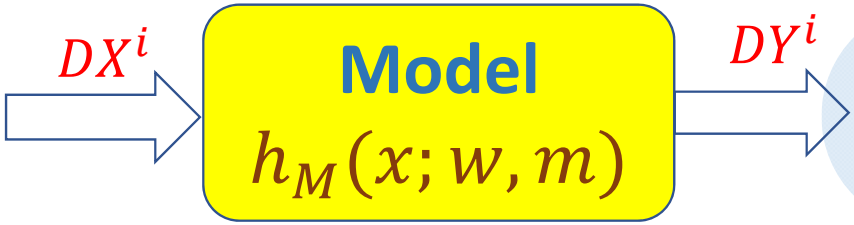
Training stage



** Clustering's object function is to minimize the maximum cluster distance.*

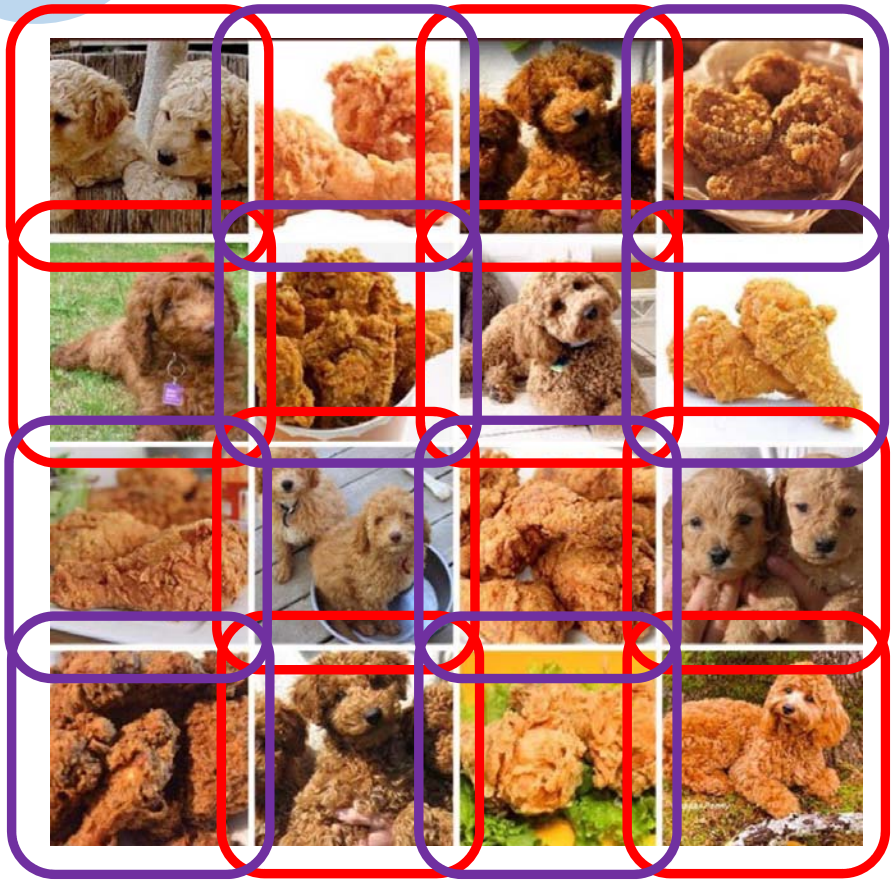


Inference stage



Unsupervised learning

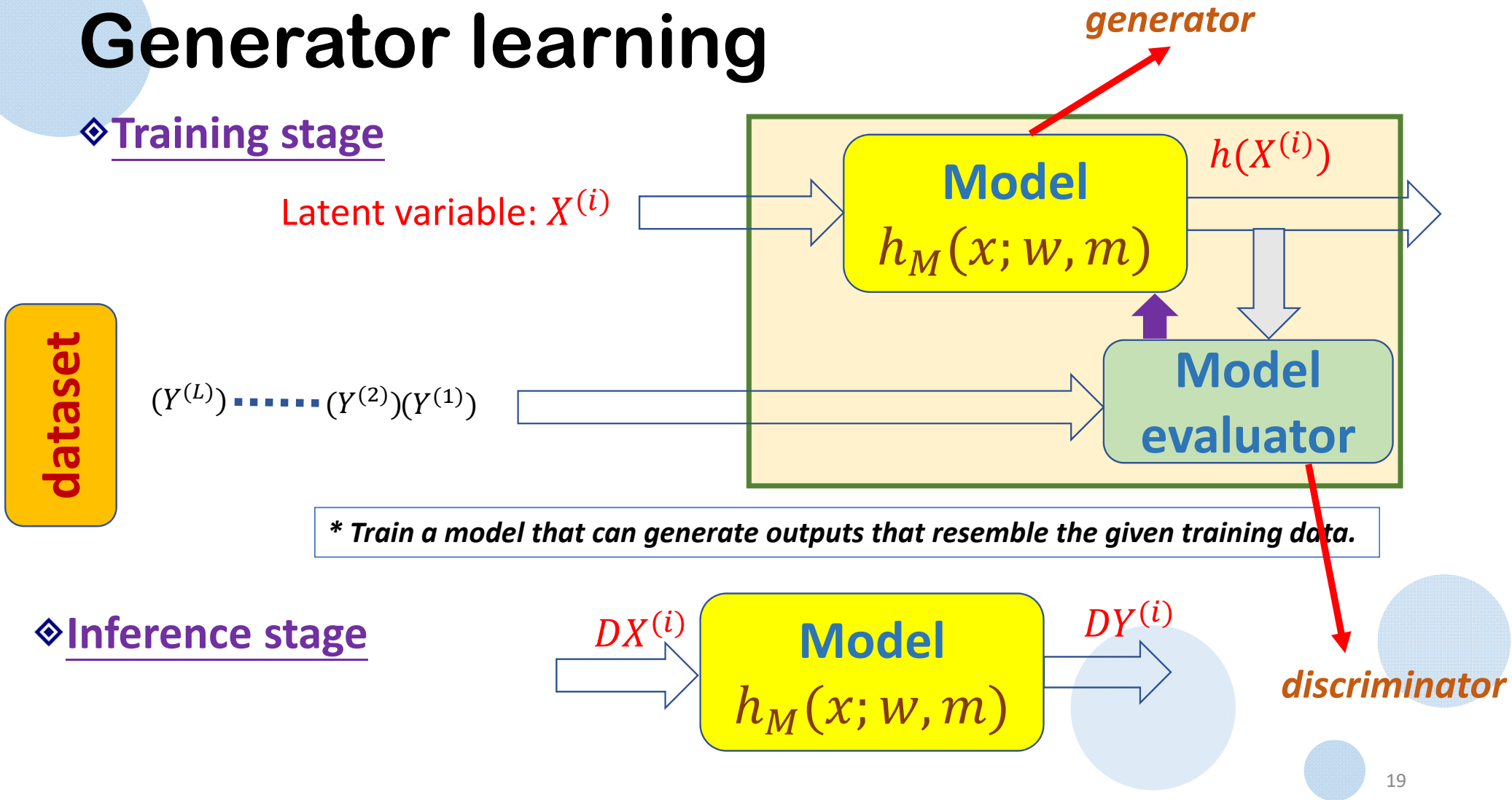
Clustering



Dog

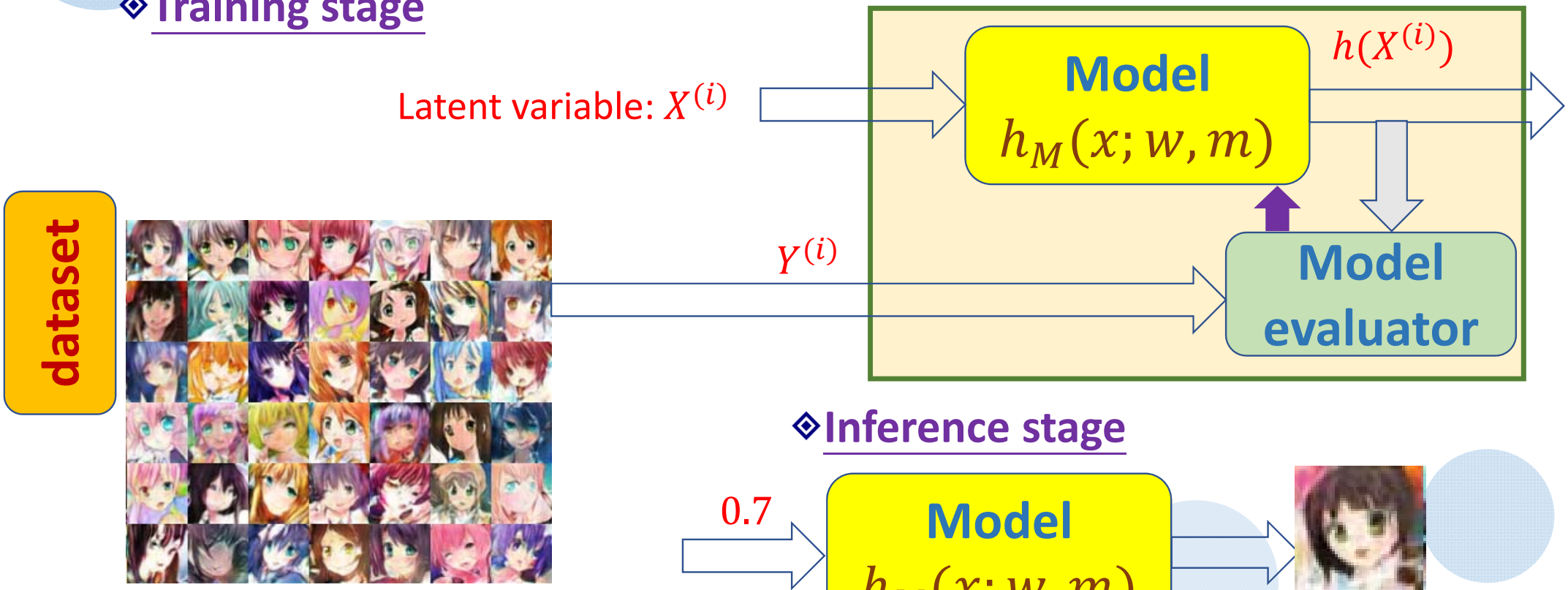
Fried Chicken

Generator learning



Generator learning

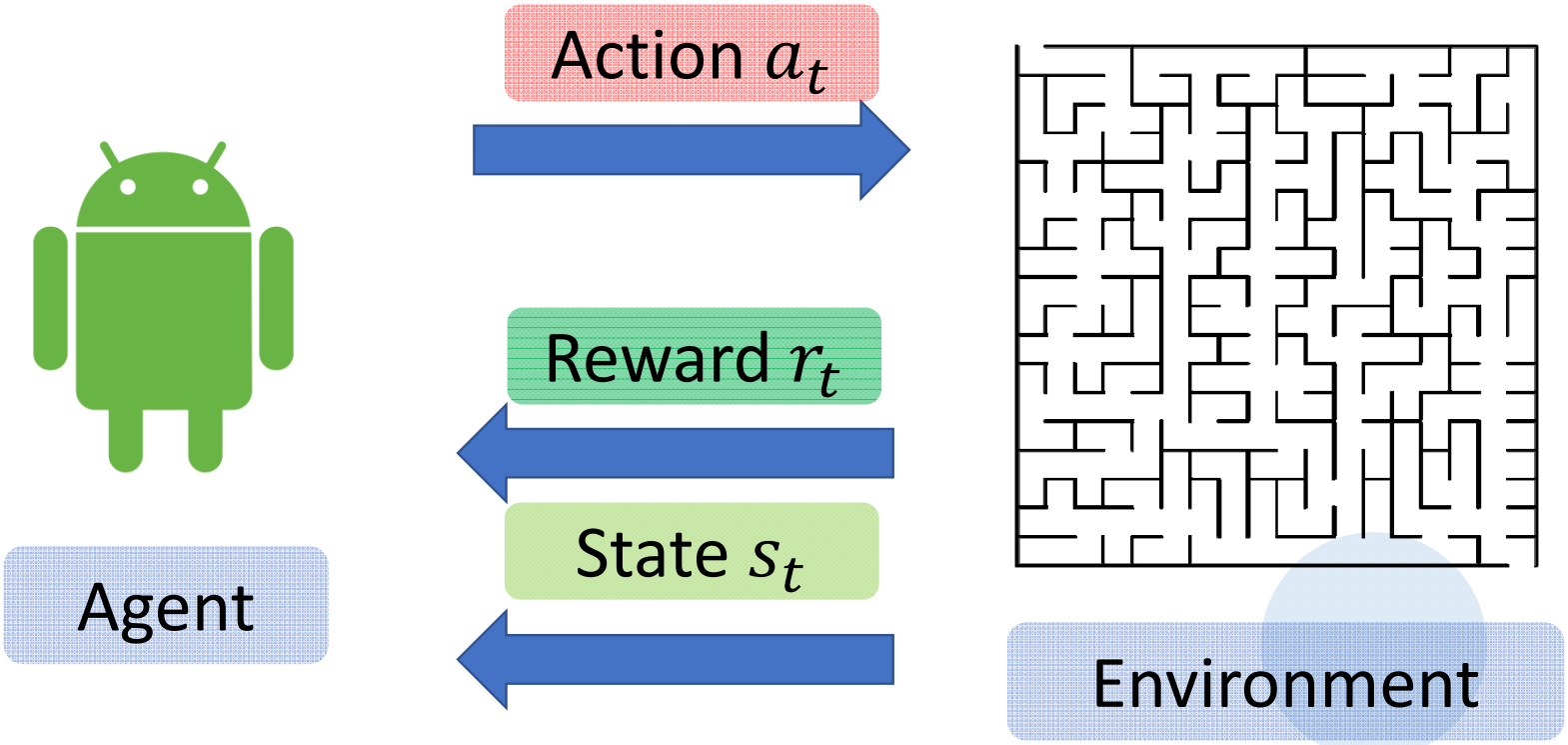
◈Training stage



◈Inference stage

Reinforcement

- Train an agent function to decide actions.



Reinforcement Learning

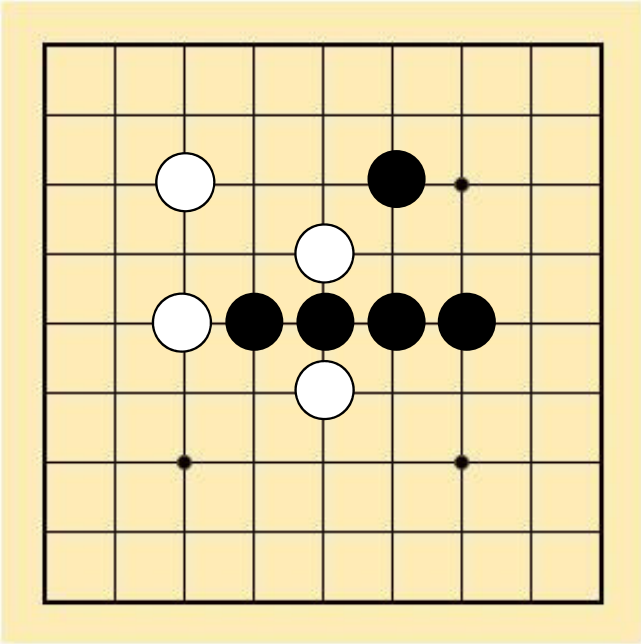
- In Gomoku, one who can form an unbroken chain of five is the winner

State s_t

Board

Environment

Opponent



Action a_t

Where to place the stone

Reward r_t

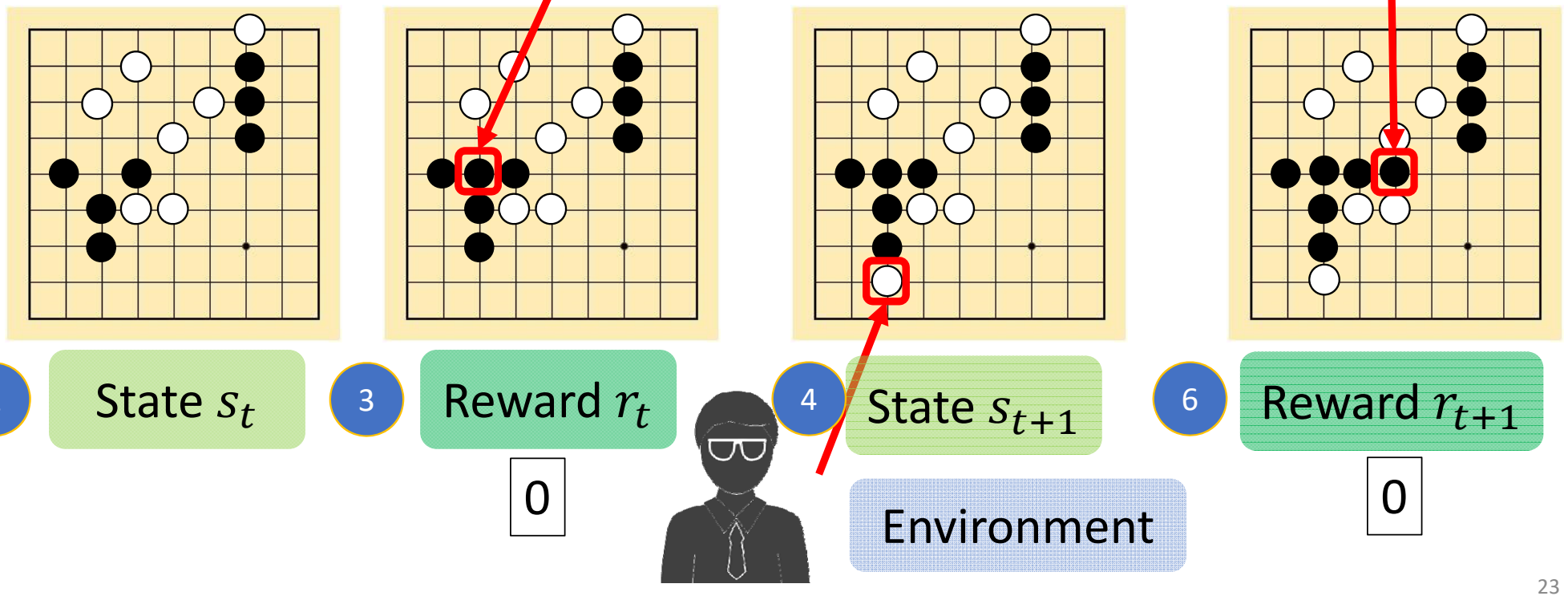
Win : get 1 point

Not-Win : get 0

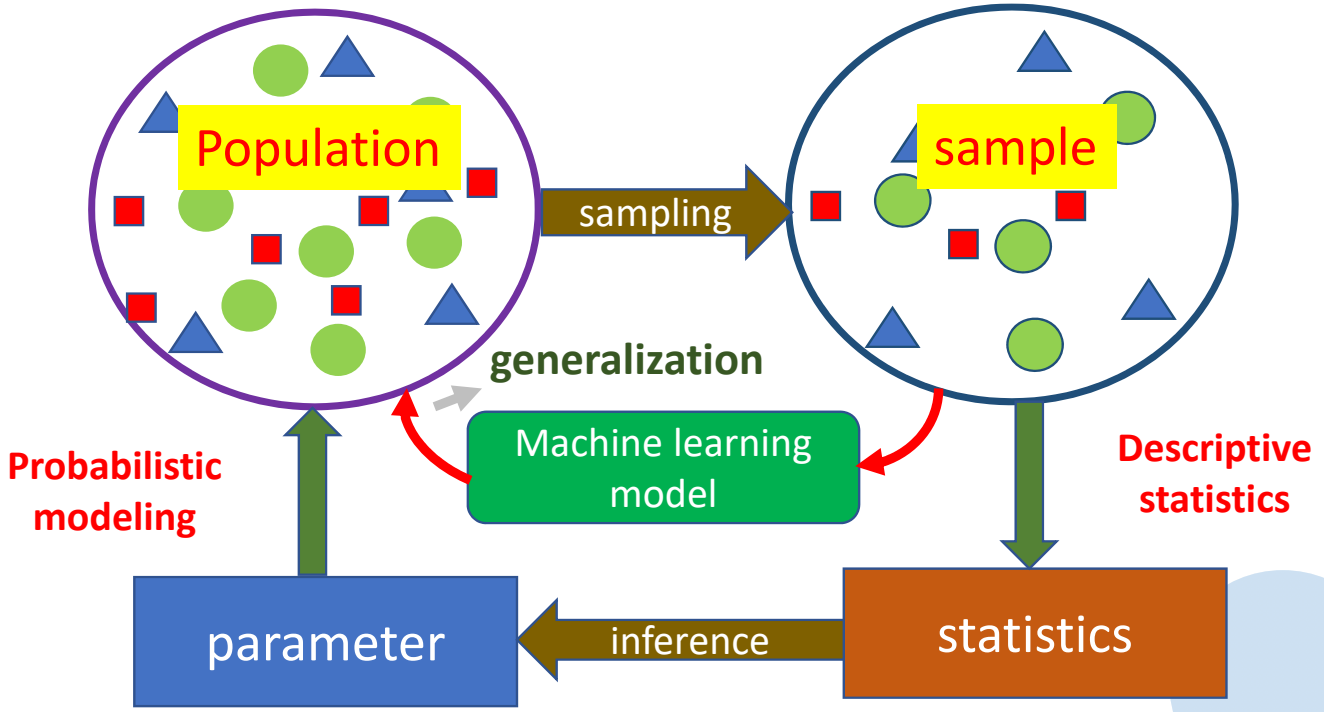
Loss: get -1 point

Reward

Agent

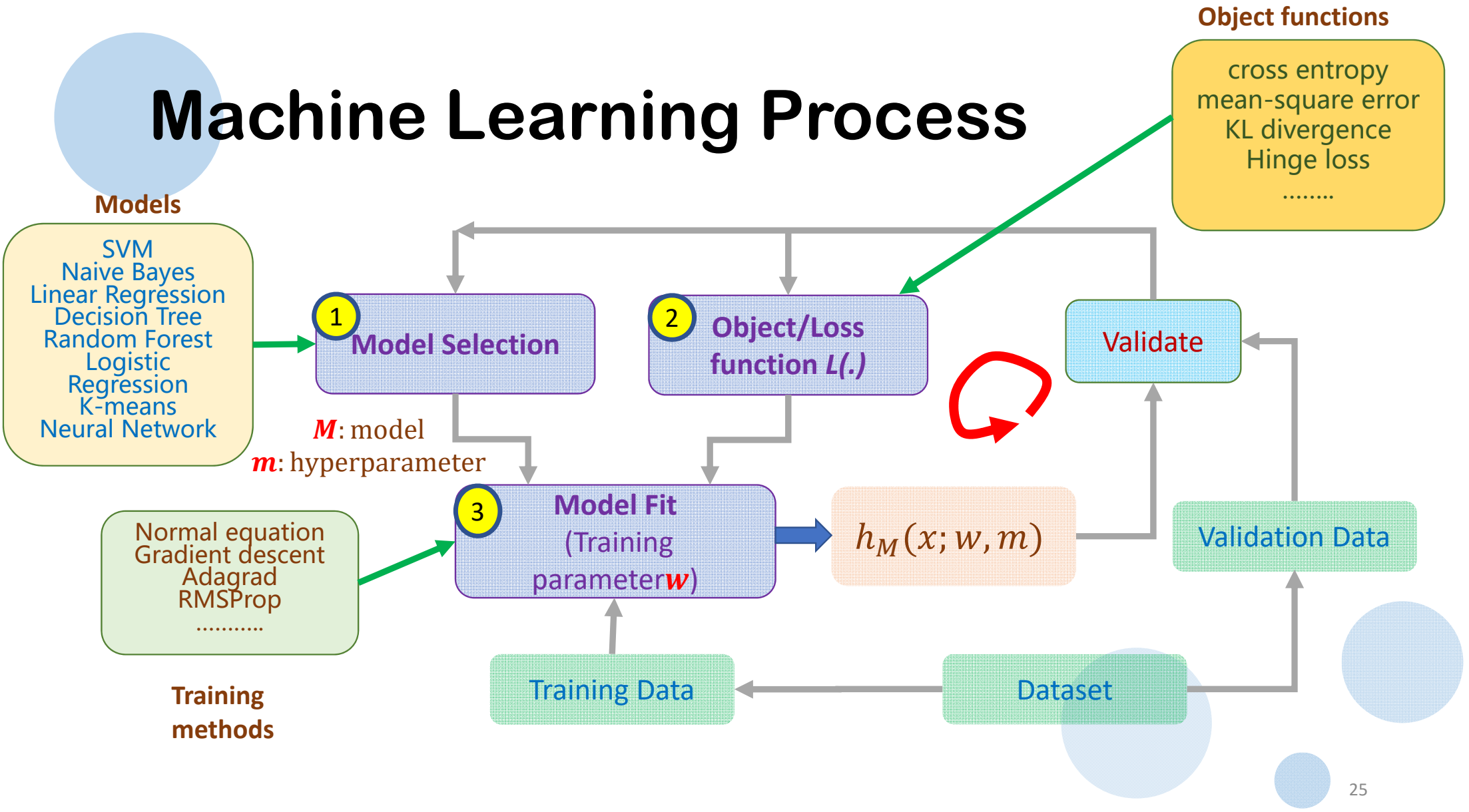


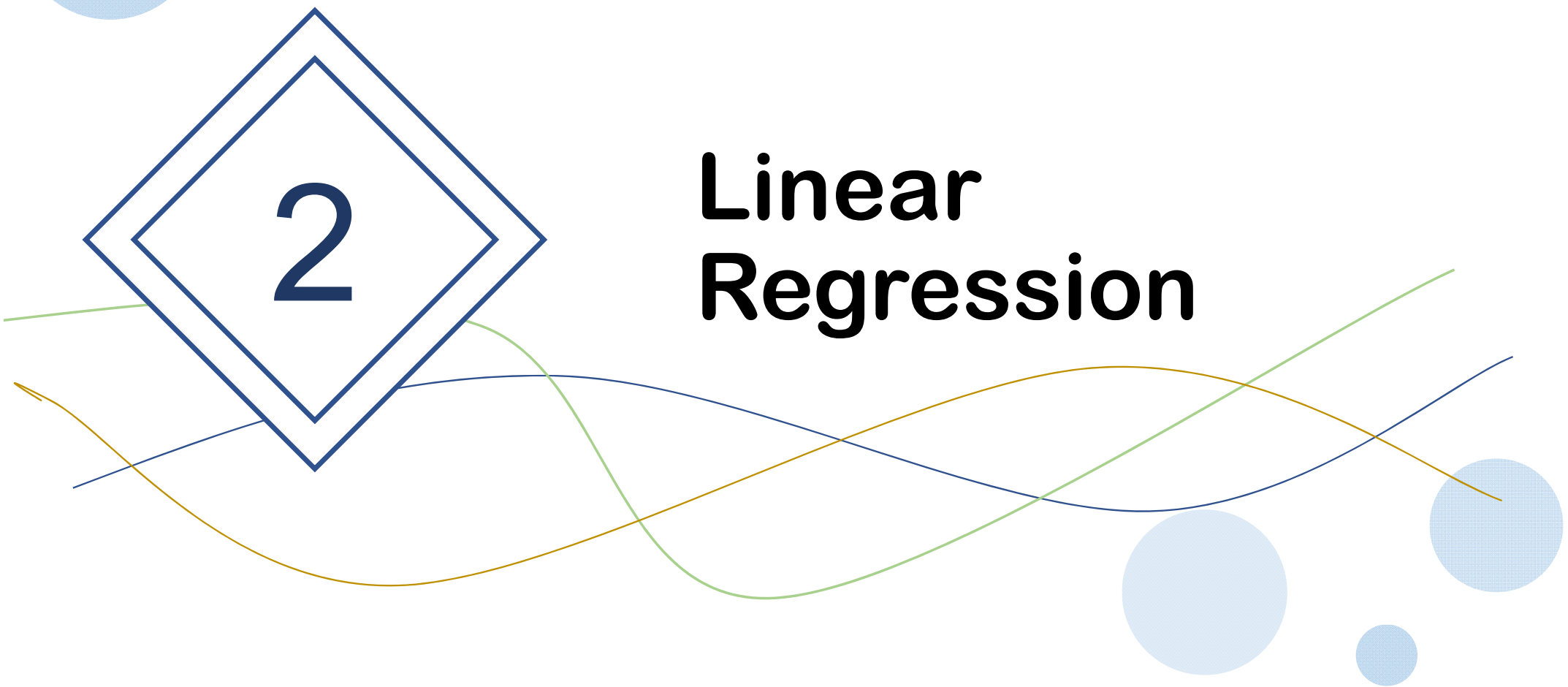
Sampling & Learning



Overfitting vs Underfitting

Machine Learning Process





Linear Regression

What's linear regression

- A *linear approach* to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables*).*
  - *Summation of weighted results*

$$y(x_1, x_2 \ldots x_n) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

Model parameter, unknown in advance.

- Some related regression terminology
  - Multiple linear regression
    - More than one independent variable
  - Multivariate linear regression
    - More than one dependent variable.
  - Polynomial regression
    - The independent variable has more than one degree in the model of linear equation

Normal equation

- A linear system of n+1 variables:

$$y(x_1, x_2 \dots x_n) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

- Suppose our dataset contains m data

$$y^{(1)} = w_0 + w_1 x_1^{(1)} + w_2 x_2^{(1)} + \cdots + w_n x_n^{(1)}$$
$$y^{(2)} = w_0 + w_1 x_1^{(2)} + w_2 x_2^{(2)} + \cdots + w_n x_n^{(2)}$$
$$\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots$$
$$y^{(m)} = w_0 + w_1 x_1^{(m)} + w_2 x_2^{(m)} + \cdots + w_n x_n^{(m)}$$

- Can be rewritten as $Xw = b$
- **w** can be solved using *normal equation*:

$$X^T X w = X^T b \Rightarrow w = (X^T X)^{-1} X^T b$$

Polynomial Curve Fitting (regression)

- Model :

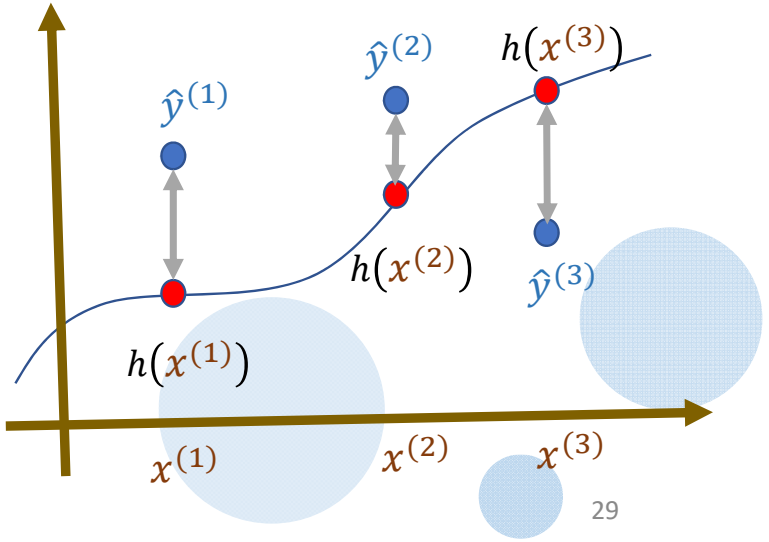
Model hyper-parameter, set in advance.

$$h(x; w, m) = w_0 + w_1 x + w_2 x^2 + \cdots w_m x^m$$

Model parameter, unknown in advance.

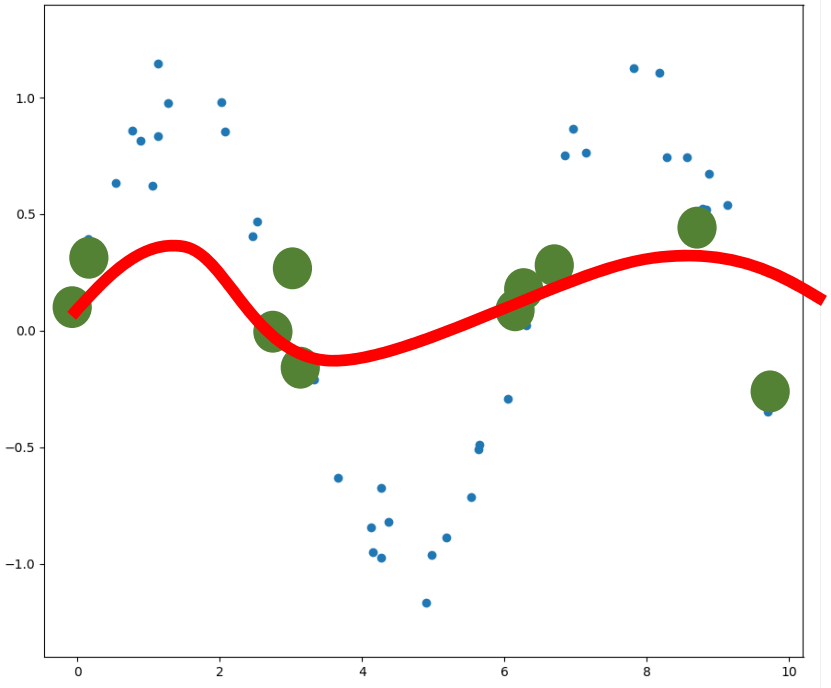
- Loss function: Mean-square-error (MSE)

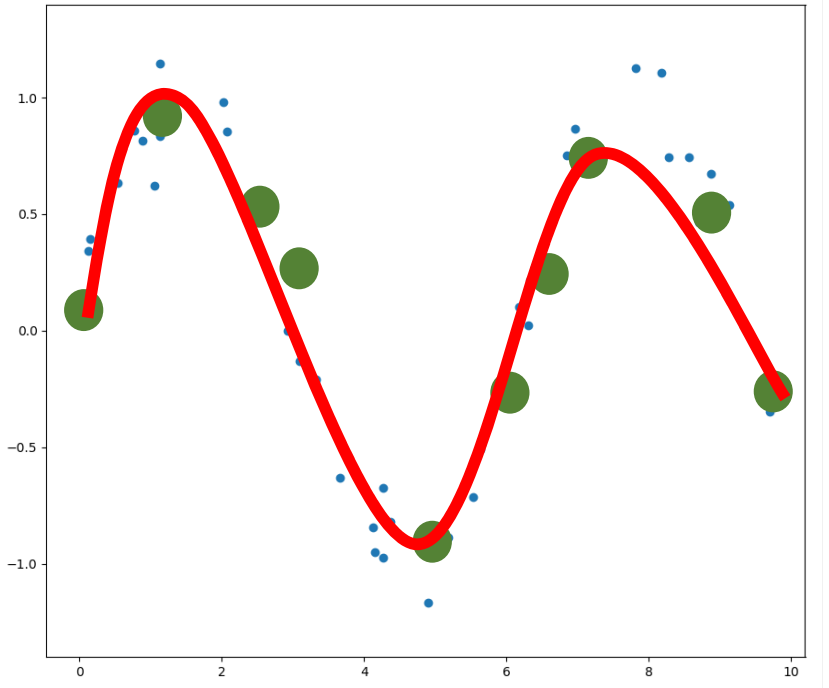
$$L(w) = \frac{1}{N} \sum_{i=1}^{N} (h(x^{(i)}; w, m) - \hat{y}^{(i)})^2$$

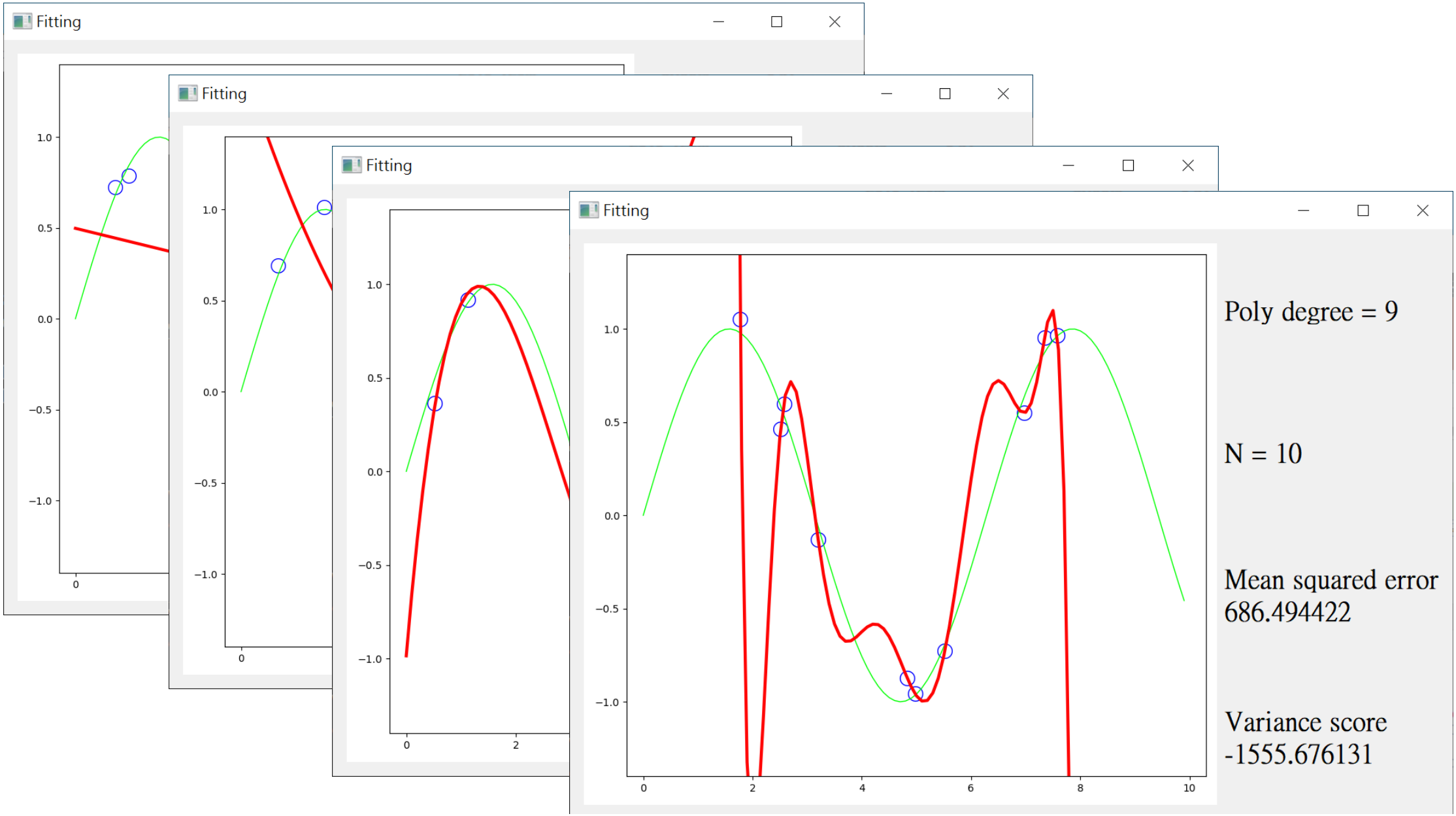


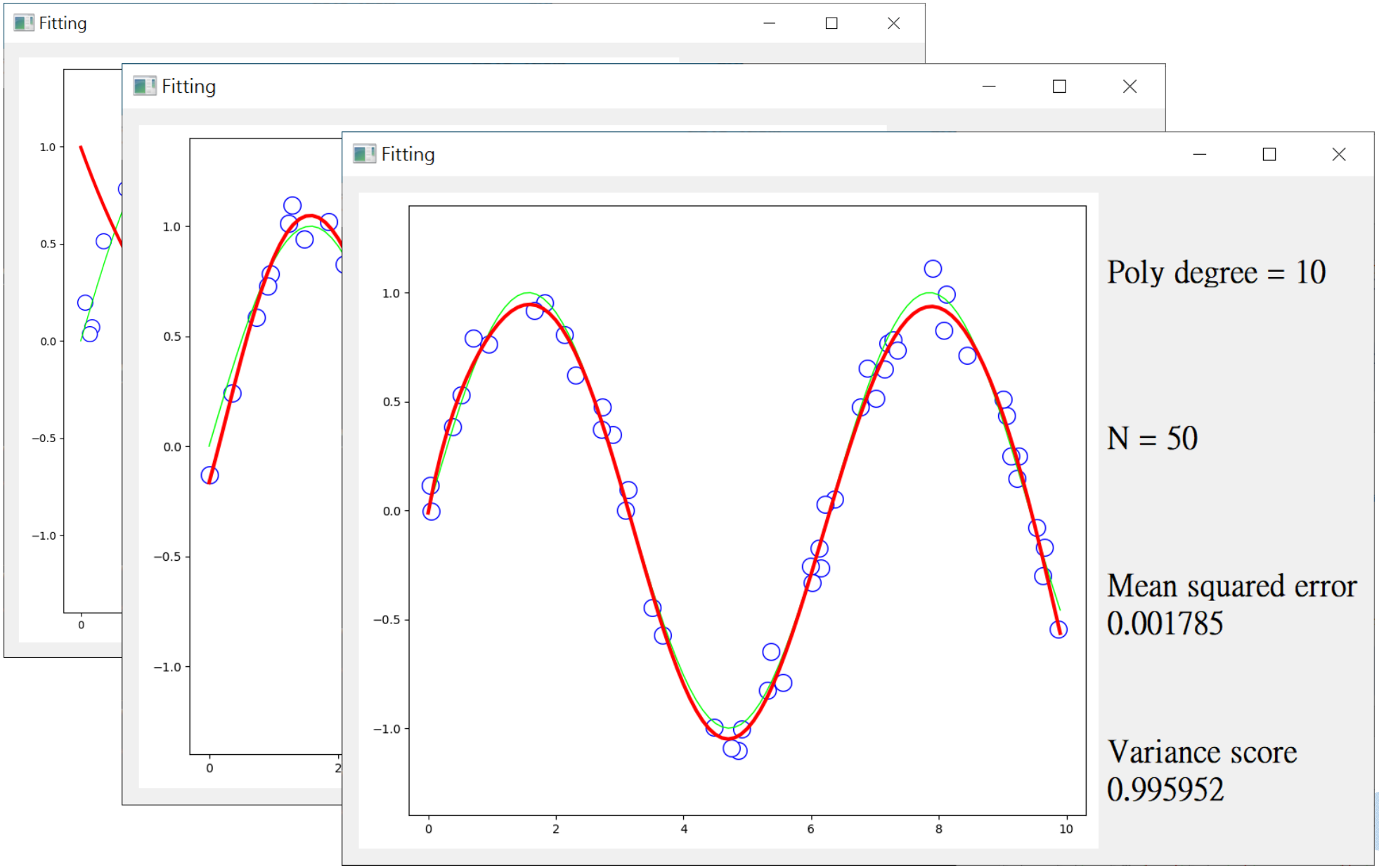
Sample size vs variance

- If the size of sample data is small, the model may vary a lot for different samples.



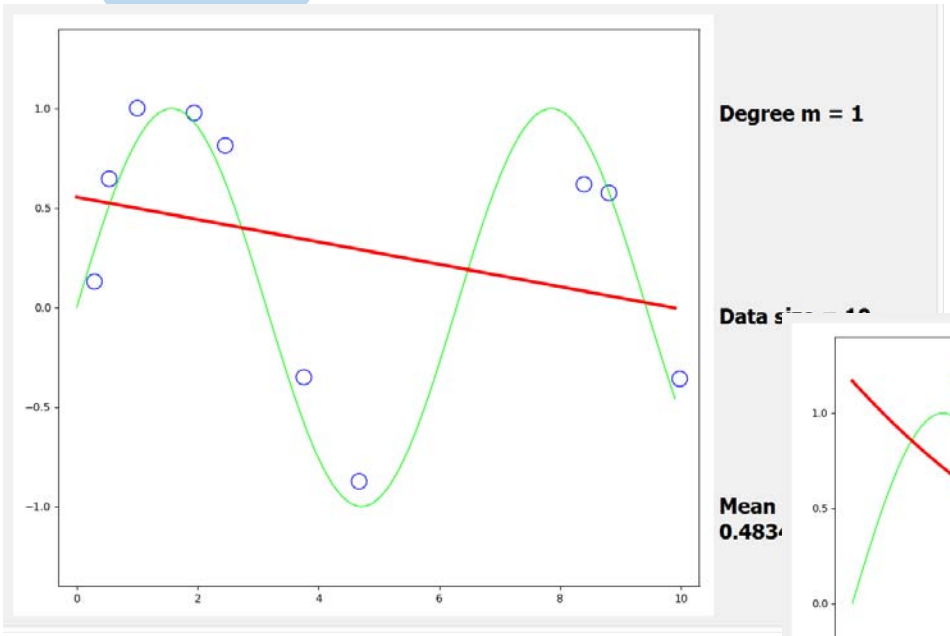


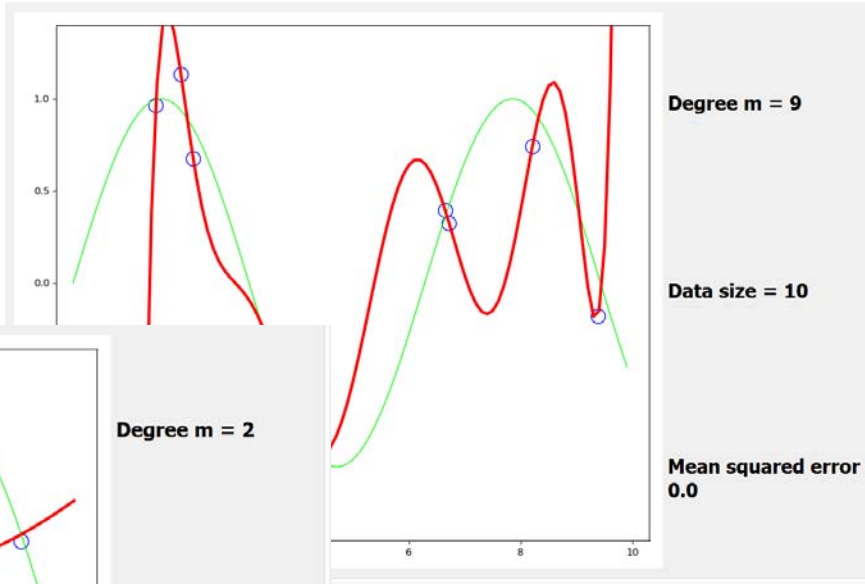

Fitting
Fitting
Fitting
Fitting
Poly degree = 9
N = 10
Mean squared error
686.494422
Variance score
-1555.676131


Fitting
Fitting
Fitting
Poly degree = 10
N = 50
Mean squared error
0.001785
Variance score
0.995952

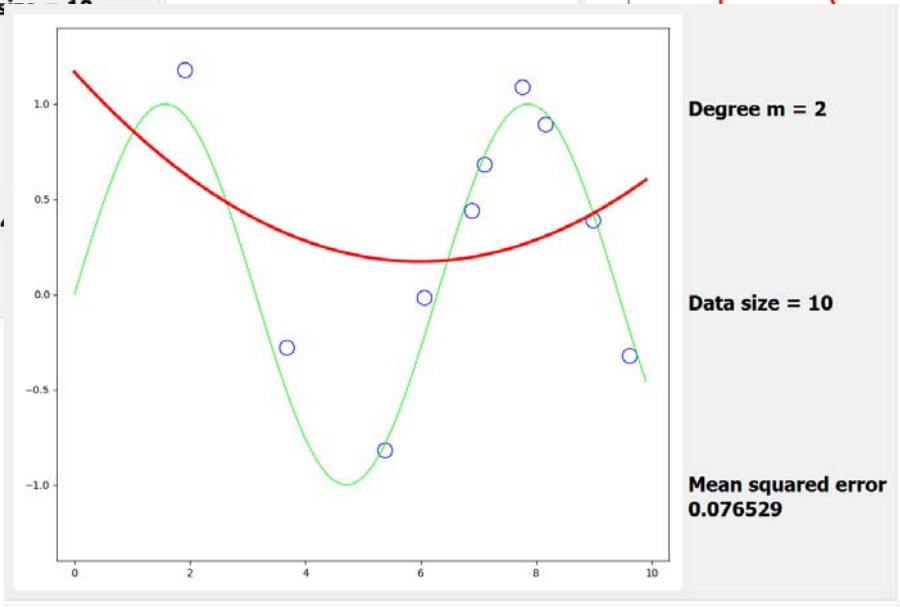
Polynomial Curve Fitting

overfitting



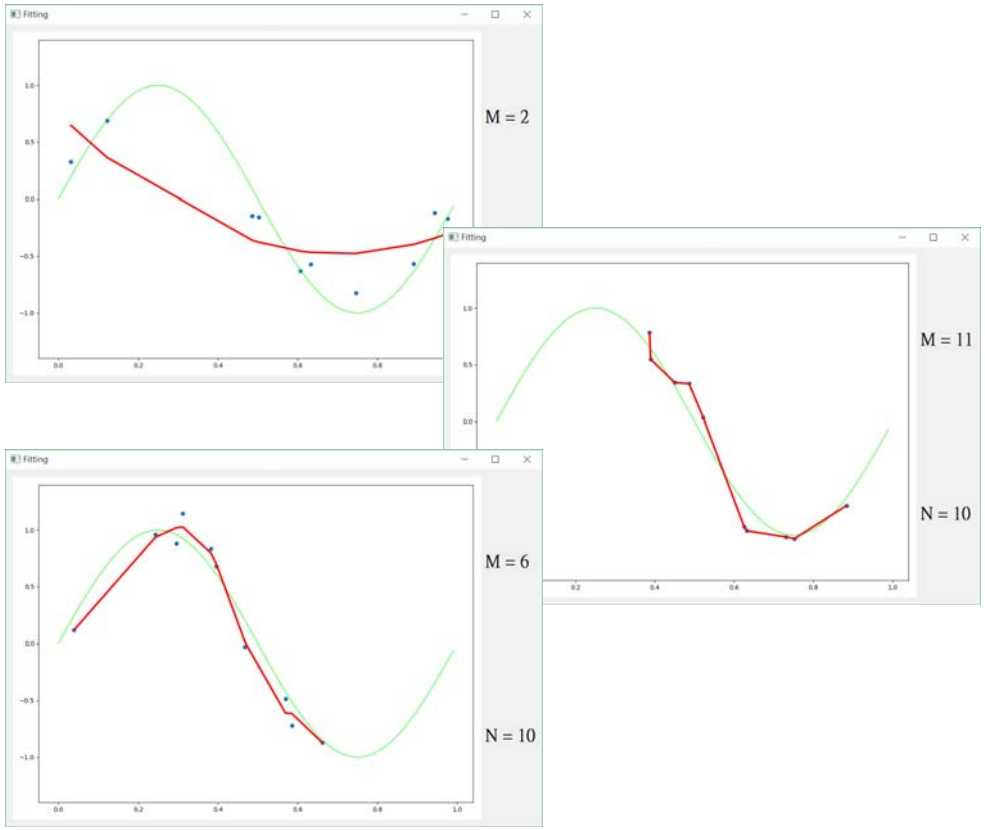


underfitting



- **Lesson:** Complex models should only be used for large data samples.

Training results of weight parameters



Weight	M = 2	M = 6	M = 11
w_o	0	0	2.67850417
w_1	-3.48815	100.8412	-776755.1085
w_2	2.46723	-1046.57	4502951.55
w_3		5108.034	-14066940.4
w_4		-12573.3	24327608.55
w_5		14976.32	-18752447.23
w_6		-6864.09	-6136446.715
w_7			20332026.69
w_8			-2160559.429
w_9			-20347291.69
w_{10}			17932831.65
w_{11}			-4913022.544

Regularization for the control of overfitting

- The coefficient α governs the relative importance of the regularization term.
- Ridge regression, L2 regularization

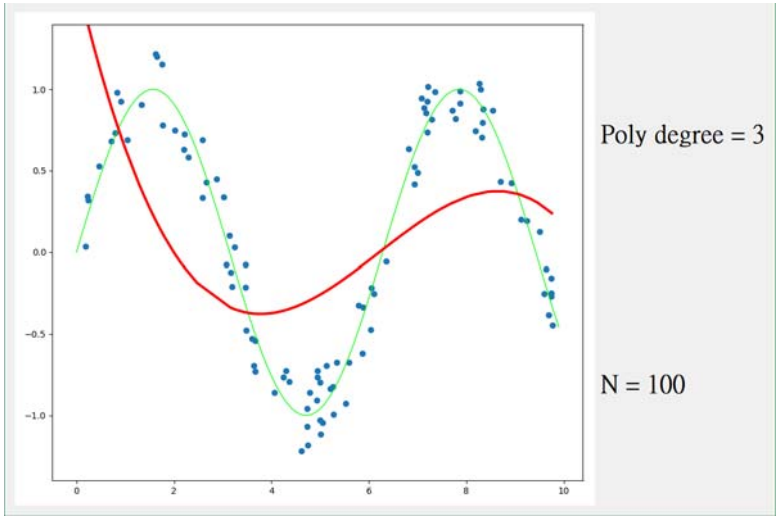
$$E(w) = \frac{1}{2}\sum_{i=1}^{N}(h(x^{(i)}; w, m) - \hat{y}^{(i)})^2 + \alpha\|w\|^2$$

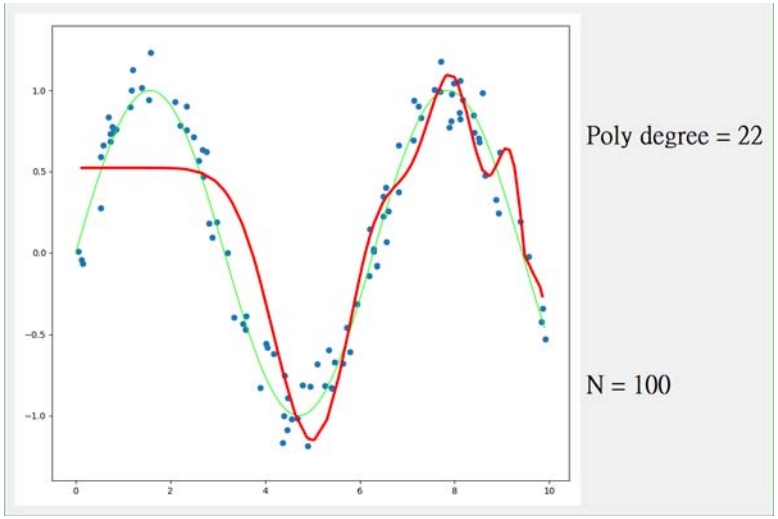
$$\|w\|^2 \equiv w^T w = w_0^2 + w_1^2 + \ \ldots \ + \ w_M^2$$

- Lasso regression, L1 regularization

$$E(w) = \frac{1}{2}\sum_{i=1}^{N}\{h(x^{(i)}; w, m) - \hat{y}^{(i)}\}^2 + \alpha\|w\|^1$$

Over-fitting and Under-fitting





Small degree (simple model)
-> underfitting
-> Less variance

Large degree (complex model)
-> overfitting
-> Less bias

Model complexity

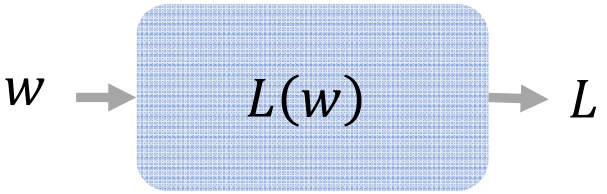
- We should choose the model with suitable complexity for the size of the given dataset.
  - High complexity -> more freedom -> low bias with potential high variance
  - Low complexity -> less freedom -> low variance with potential high bias

- Should we use all the features in the given dataset?

How to solve function's minimum/maximum

- Suppose we want to find the parameter w to minimize $L(w)$
  - For example: $\mathrm{L(w)} = \frac{1}{N}\sum_{i=1}^{N}\{h(x^{(i)}; w, m) - \hat{y}^{(i)}\}^2$
  - The most direct naïve approach is to find many w candidates.
  - What if we have multiple parameters to solve?

$$L(w_1, w_2, w_3, \cdots, w_{100})$$



    - the number of candidates will be very huge. Ex: 10^{100}
  - The actual weights to train could be up to millions.

Gradient Descent Optimization Method

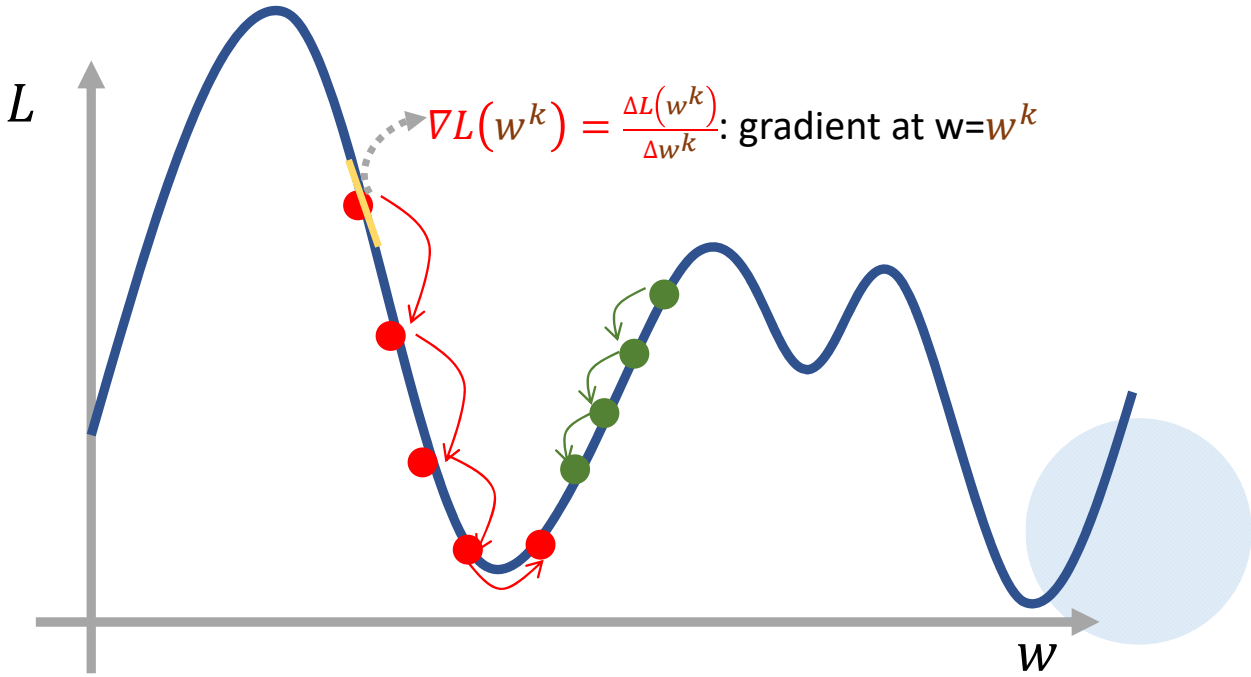
for k in range (max_iteration):

$w^{k+1} = w^k - \eta \nabla L(w^k)$ # update weight

if $\left\| w^{k+1} - w^k \right\| < \epsilon$ # stopping criteria

break

η : *adjustment (learning) rate*



- If the increase of W can reduce the loss, W will be increased.

Gradient Descent

Training data: $(x^{(i)}, \hat{y}^{(i)})$

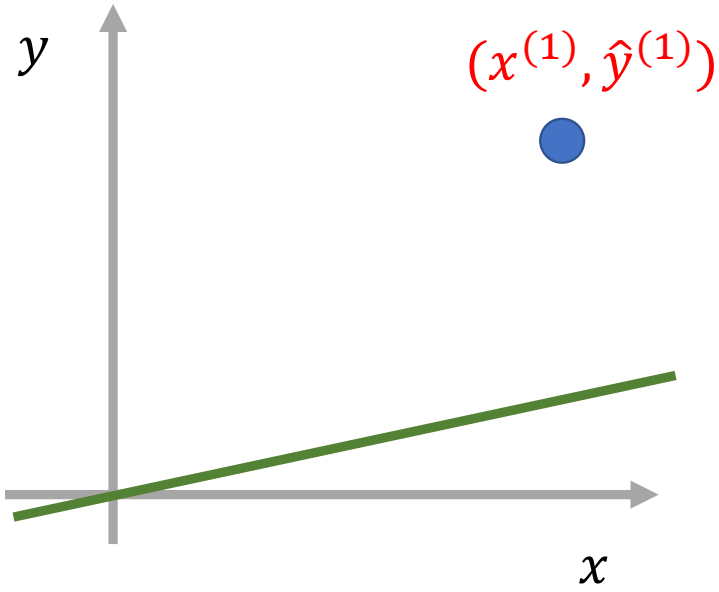
$$h(x; w, m) = x \times w$$

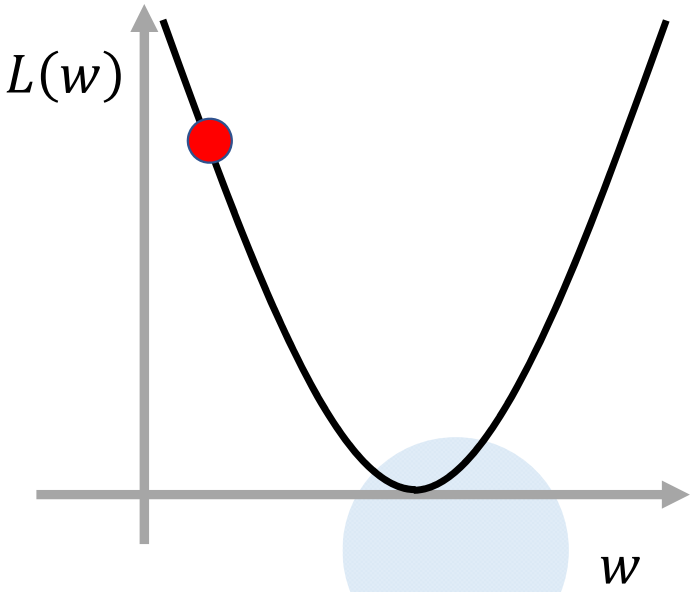
$$\mathrm{L(w)} = (x^{(1)} \times w - \hat{y}^{(1)})^2$$

Model

Epoch = 0

Loss



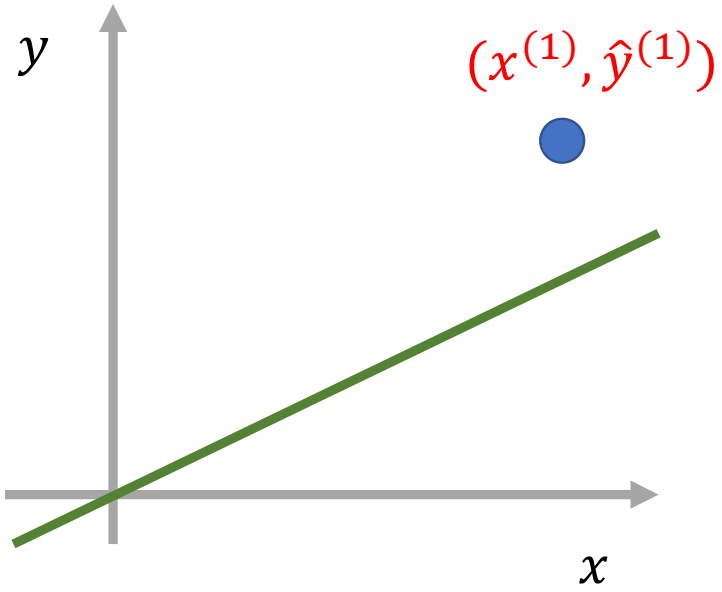


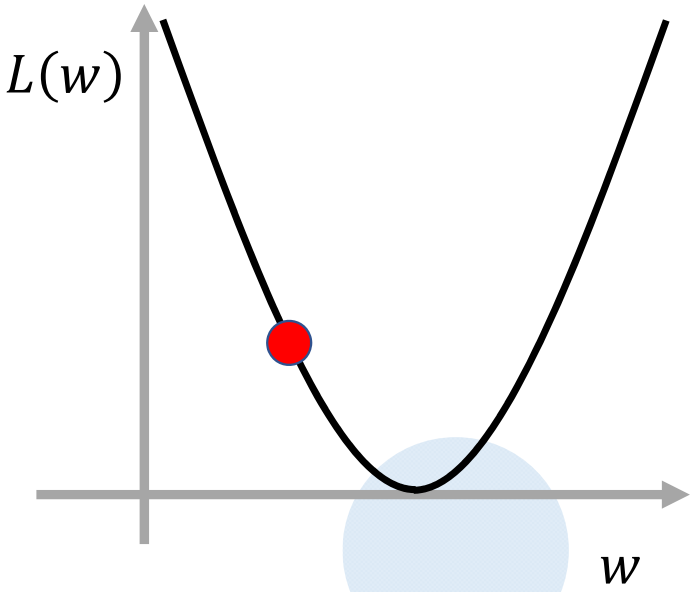
Gradient Descent

Model

Epoch = 10

Loss



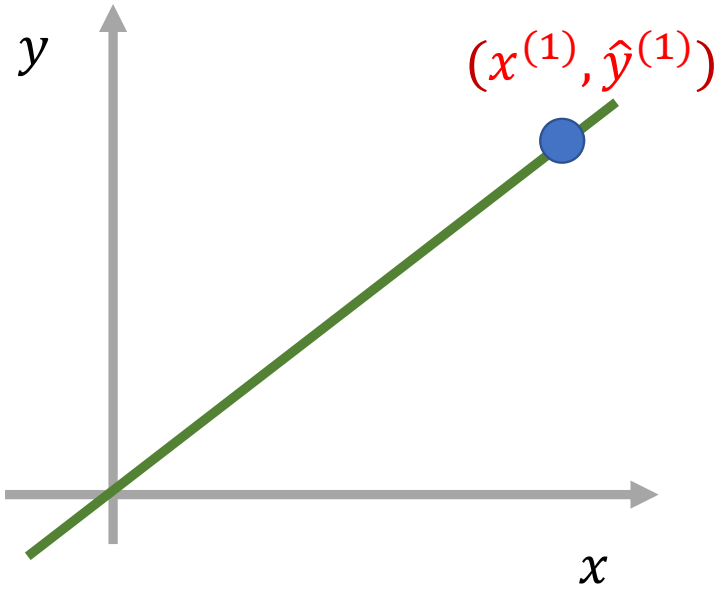


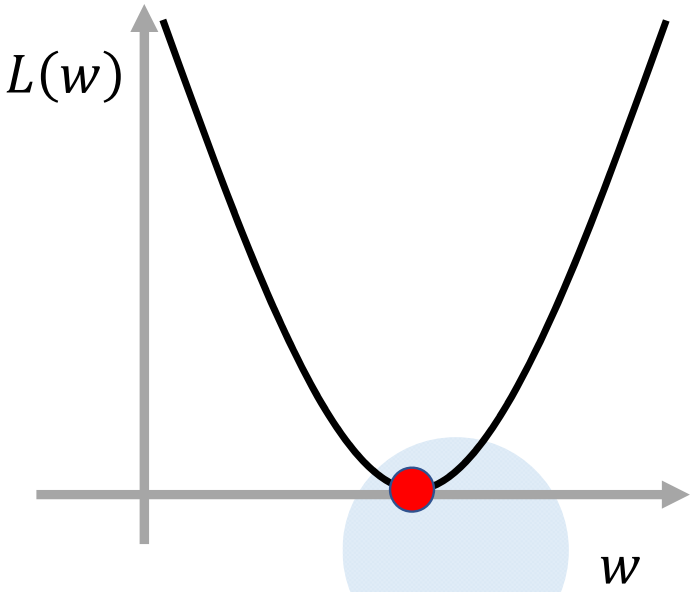
Gradient Descent

Model

Epoch = 100

Loss





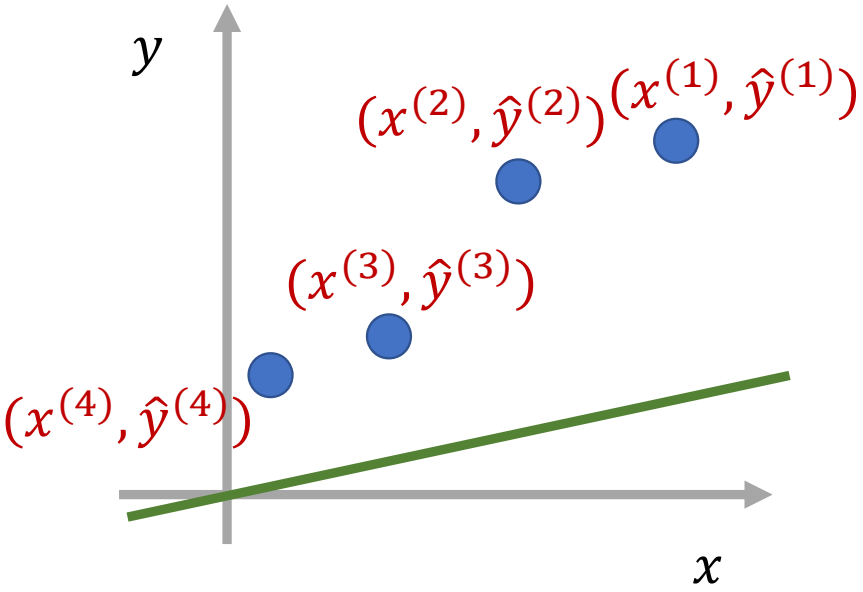
Gradient Descent

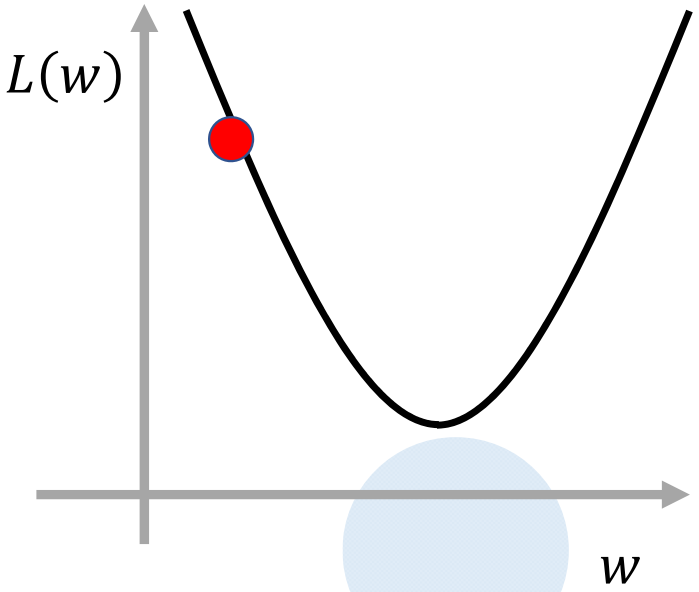
$$L(w) = \sum_{i=1}^{4} (x^{(i)} \times w - \hat{y}^{(i)})^2$$

Model

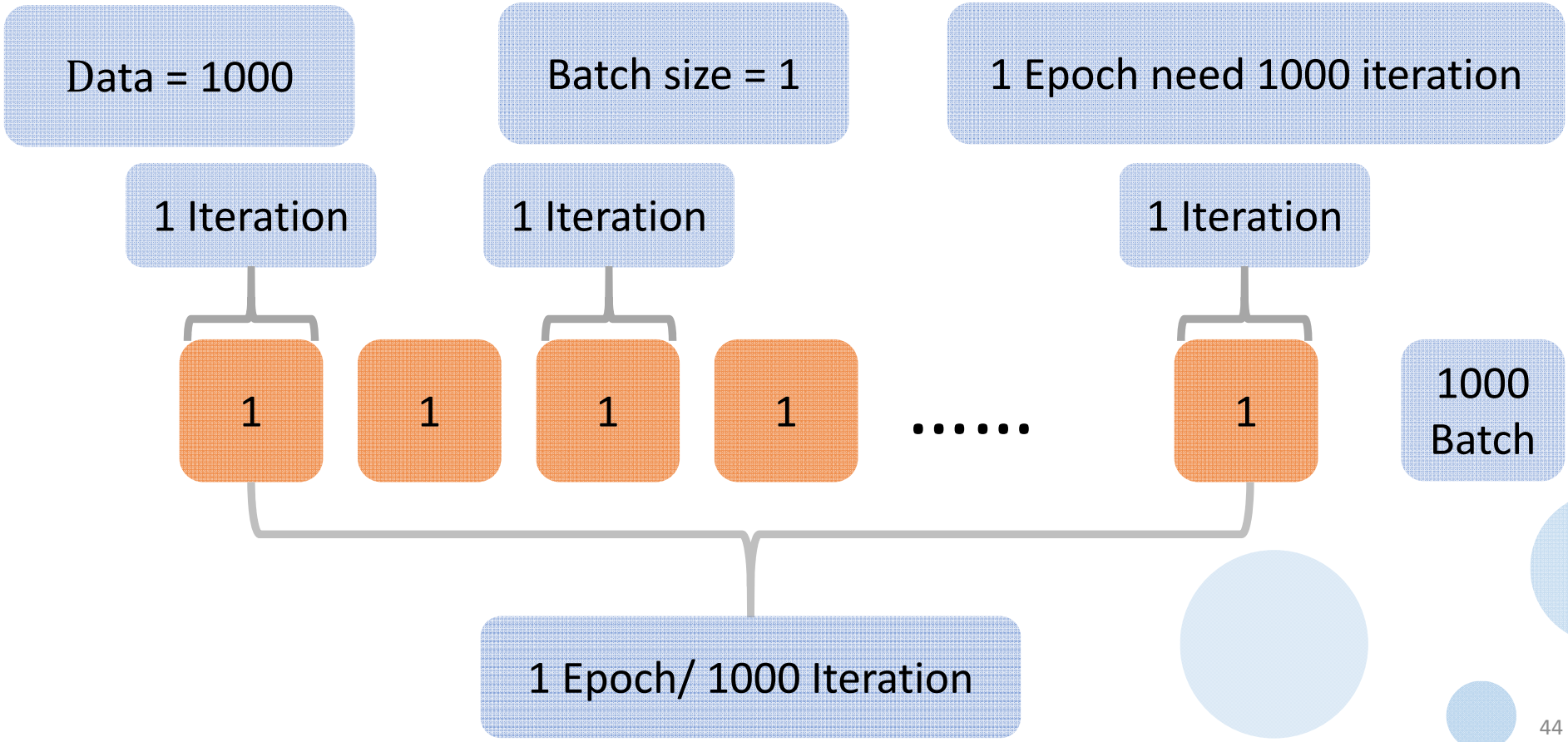
Epoch = 0

Loss

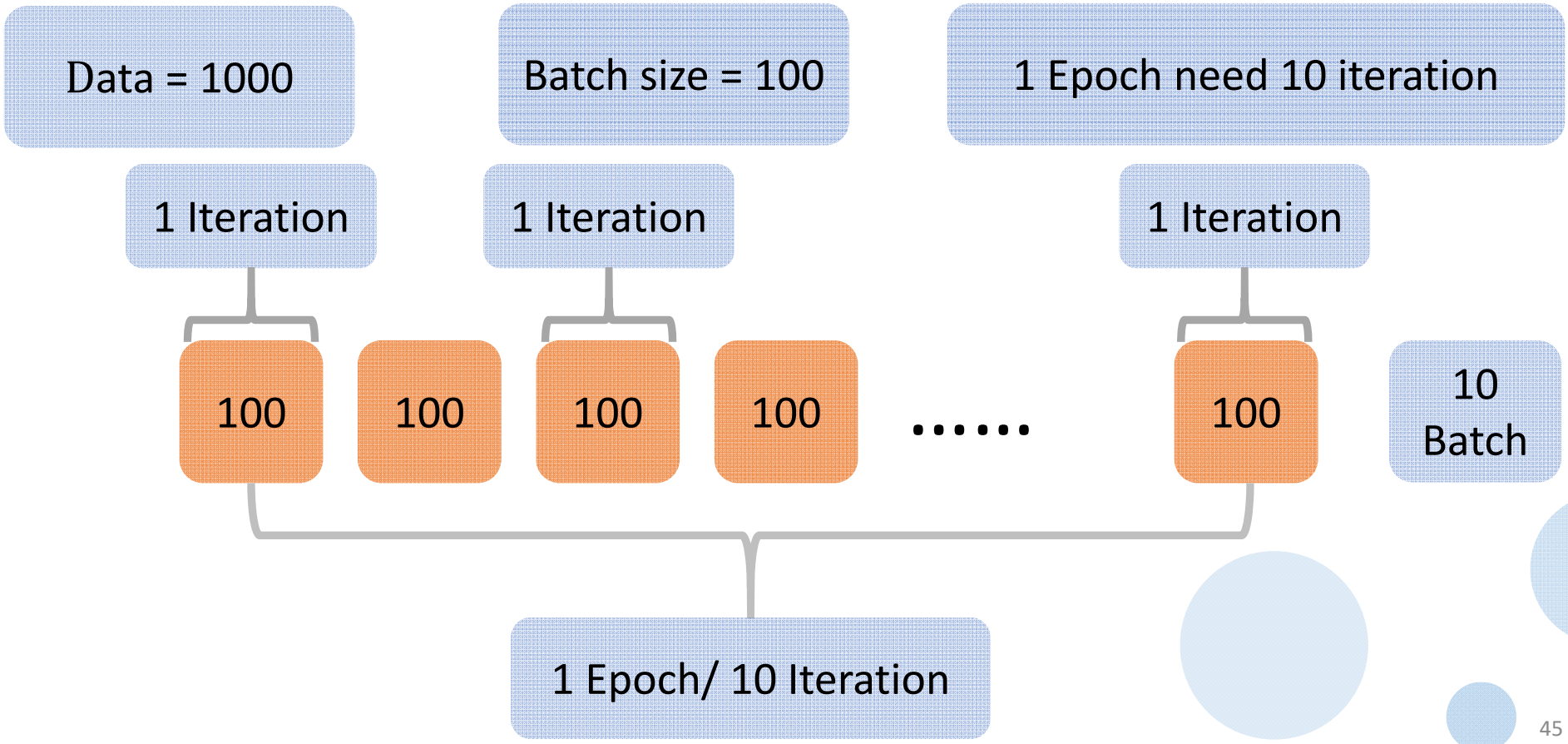




Batch, Epoch, Iteration

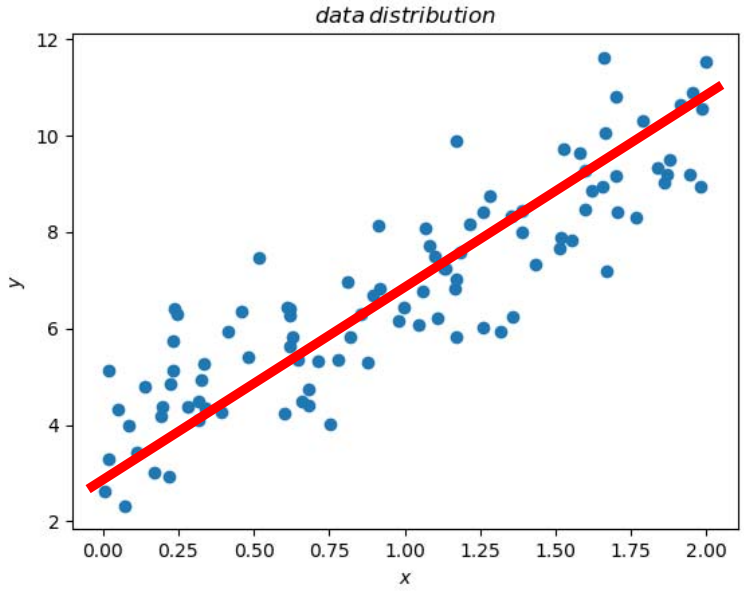


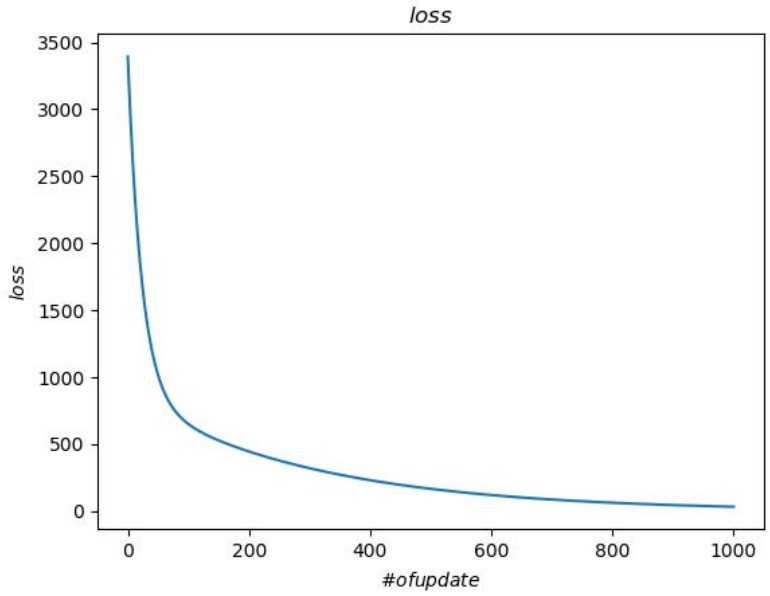
Batch, Epoch, Iteration



Linear Regression Training process

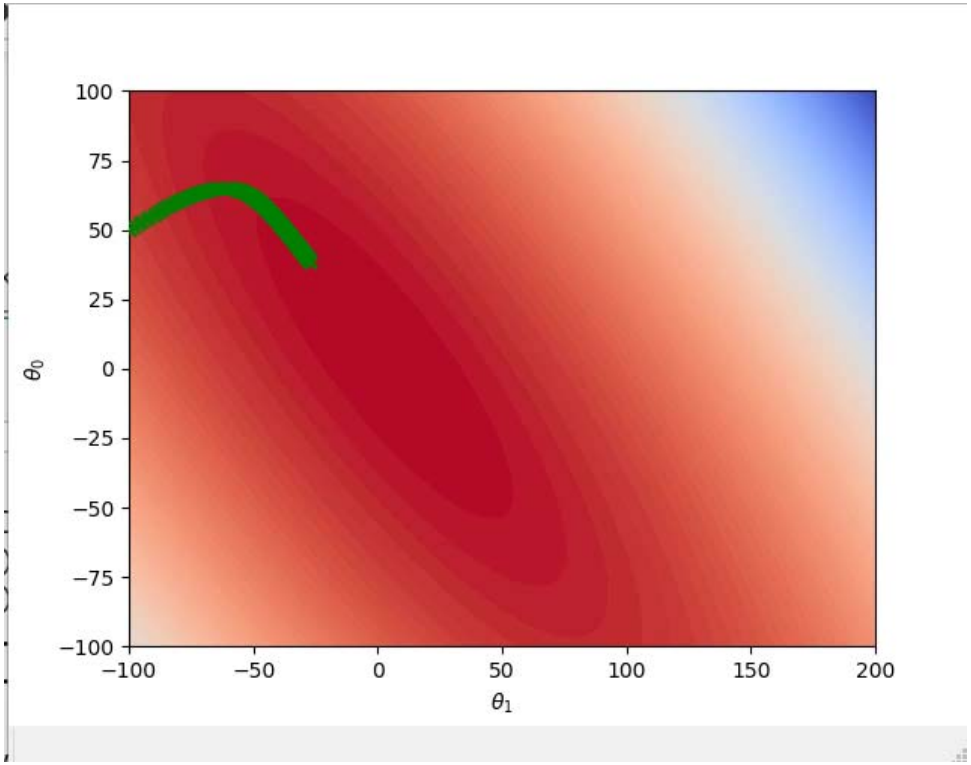
- Data and loss

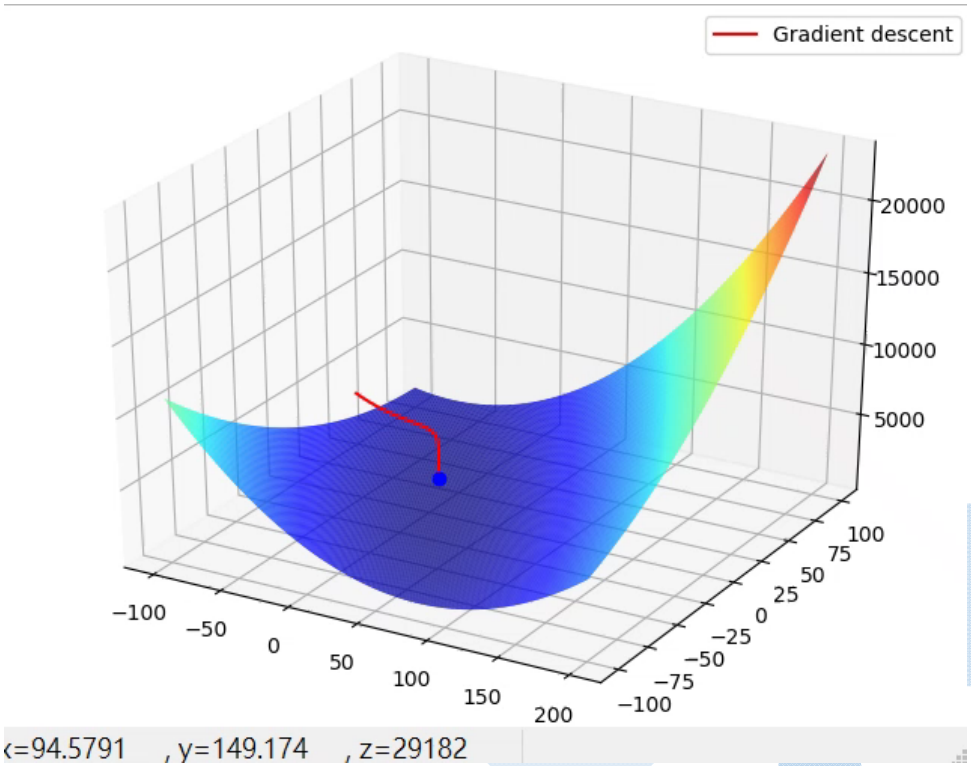




Linear Regression Training process

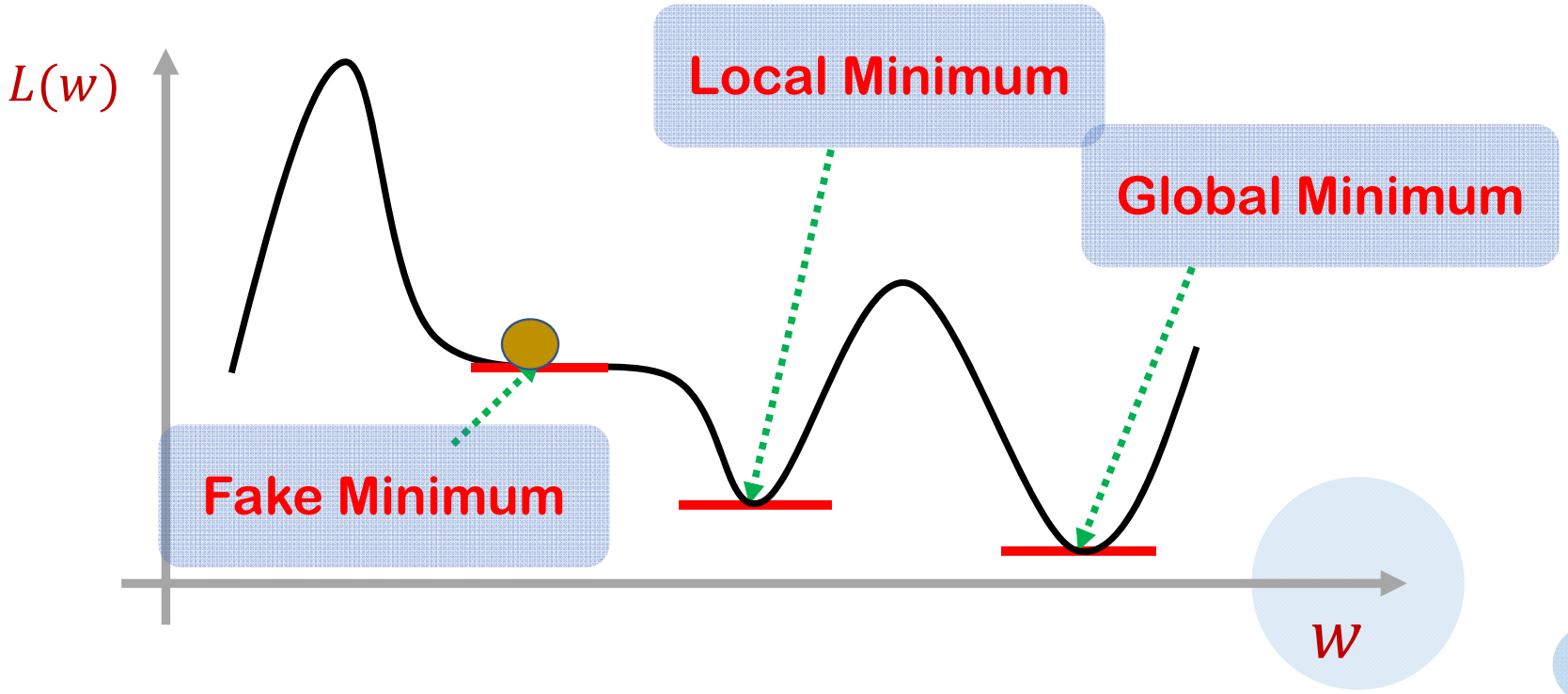
- Training process



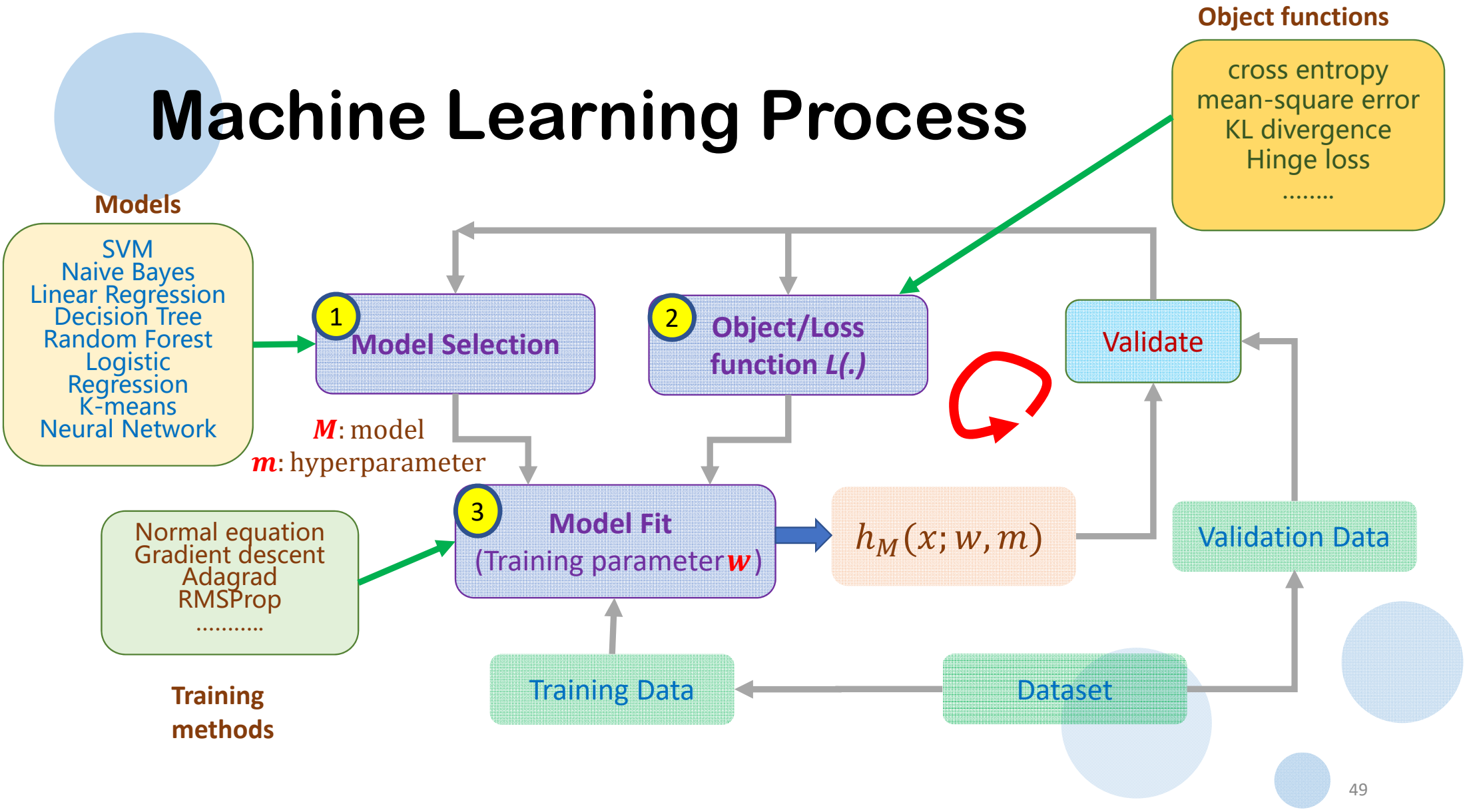


Problems of Gradient Descent

- Stuck at the local minimum points.
- Oscillate around the minimum point.
- Stuck at the saddle points.
- Very slow update at plateau.



Machine Learning Process



Ex & Hyper (meta) -parameters

Setting of sklearn decision-tree module

sklearn.tree.DecisionTreeClassifier

class sklearn.tree. **DecisionTreeClassifier** (*criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False*)

[source]

Setting of sklearn SVM module

sklearn.svm.SVC

class sklearn.svm. **SVC** (*C=1.0, kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None*)

[source]

Import package

Load dataset

Choose model
Set hyper para

Set label

Fit model

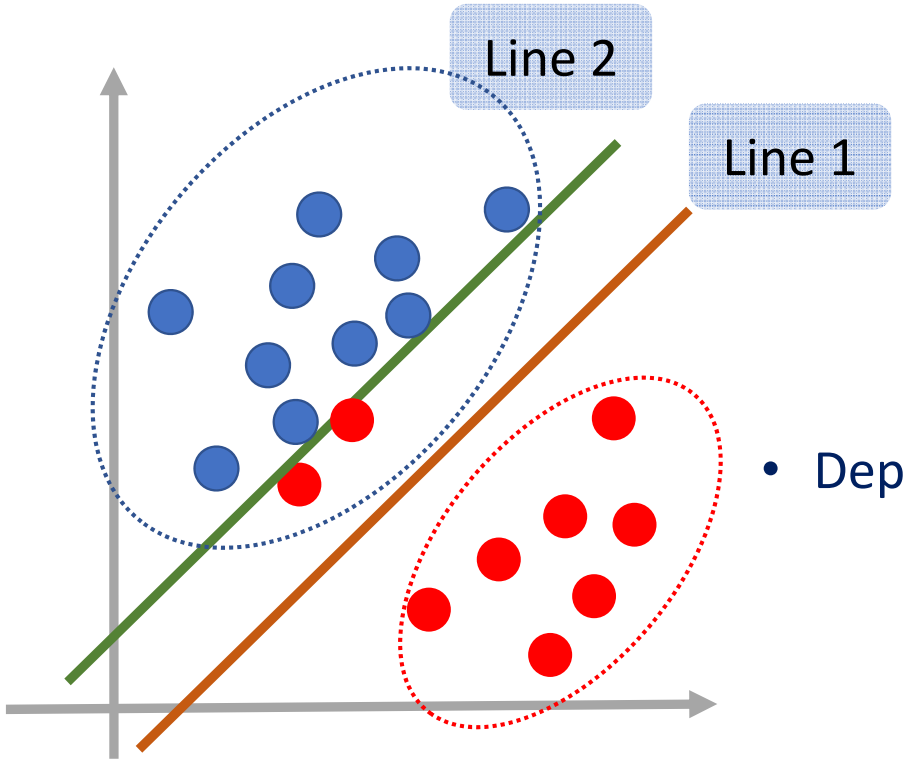
inference

Display result

```
from sklearn import svm
from sklearn import datasets

#digit dataset from sklearn
digits = datasets.load_digits()

#creat the LinearRegression  model
clf = svm.SVC(C=1.0, kernel='linear')

#set training set
x, y = digits.data[:-1], digits.target[:-1]

#train model

clf.fit(x,y)

#predit
y_pred = clf.predict([digits.data[-1]])
y_true = digits. target[-1]

print(y_pred)
print(y_true)
```

```
[8]
8
```

Which separation line is better for binary classification ?

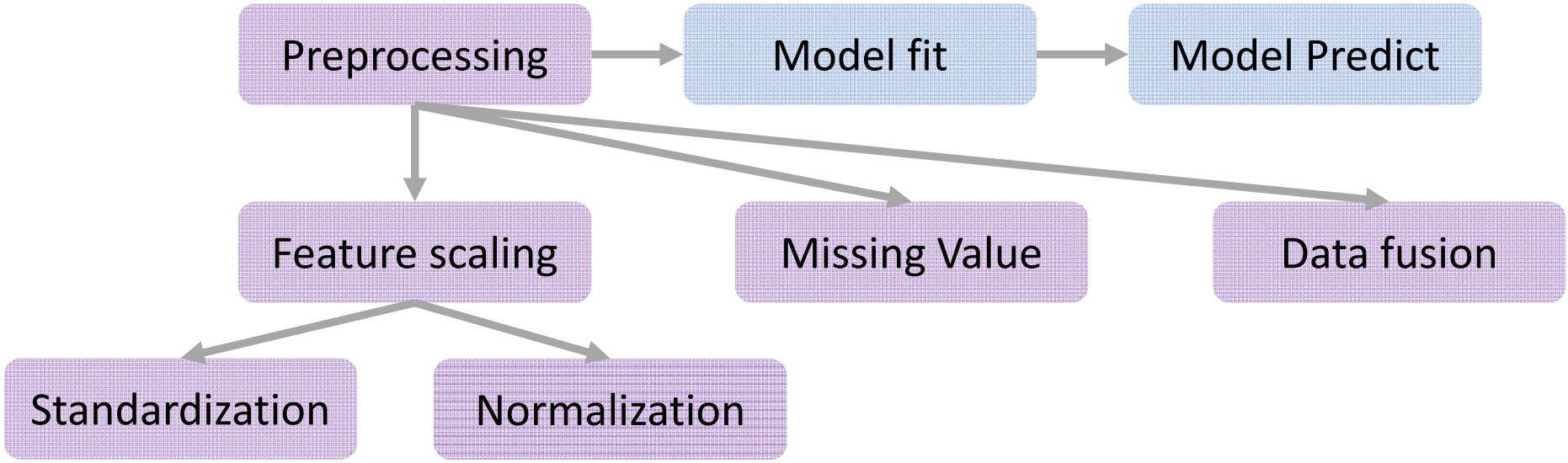


- Depend on your defined **goodness**

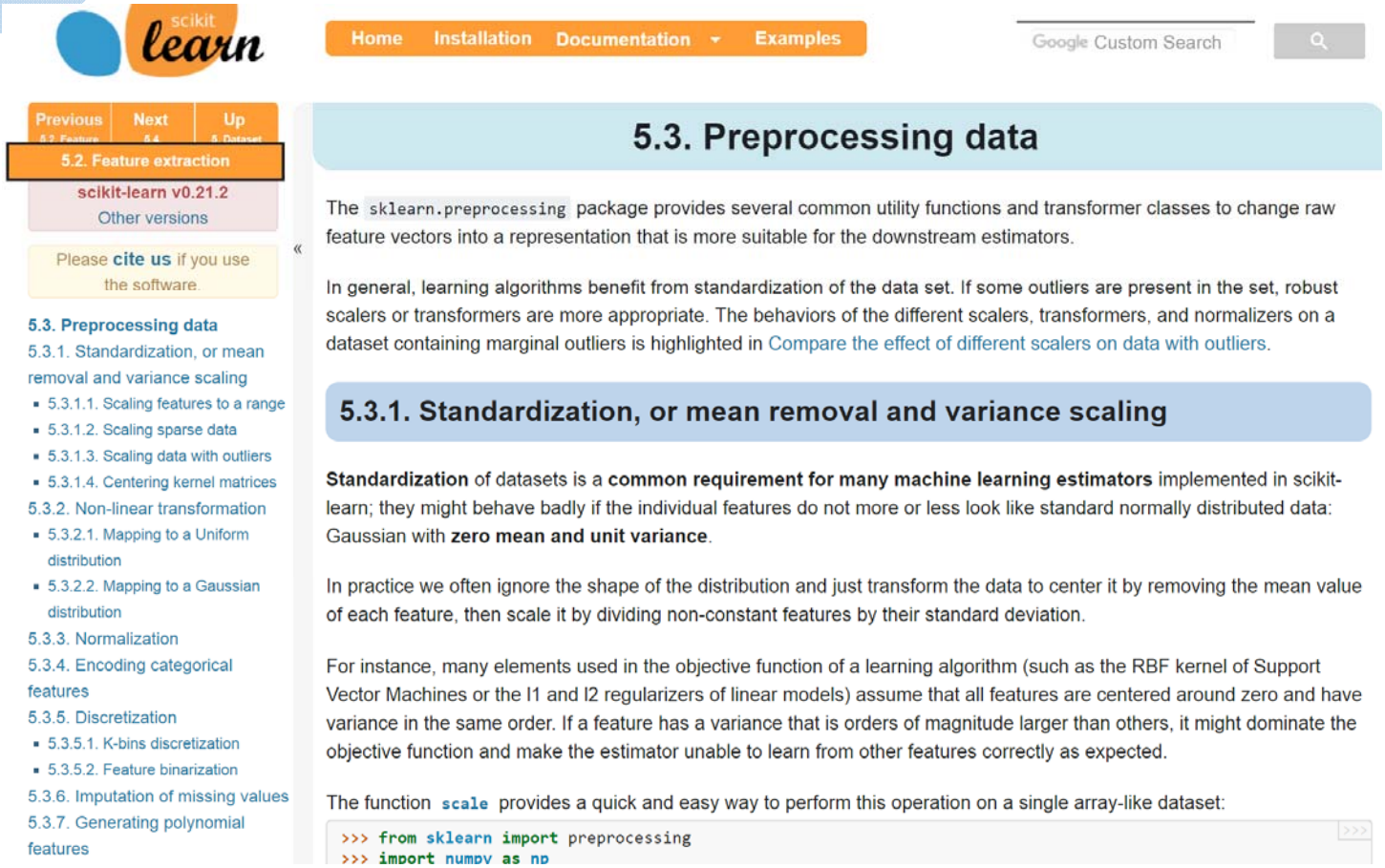
- Think about how the clustering approach (unsupervised) classifies.

Data preprocessing

Preprocessing before training



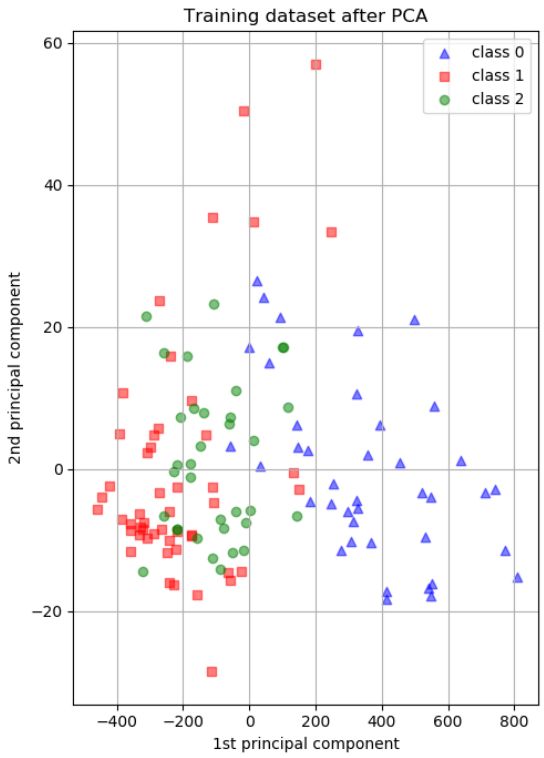
Preprocessing package

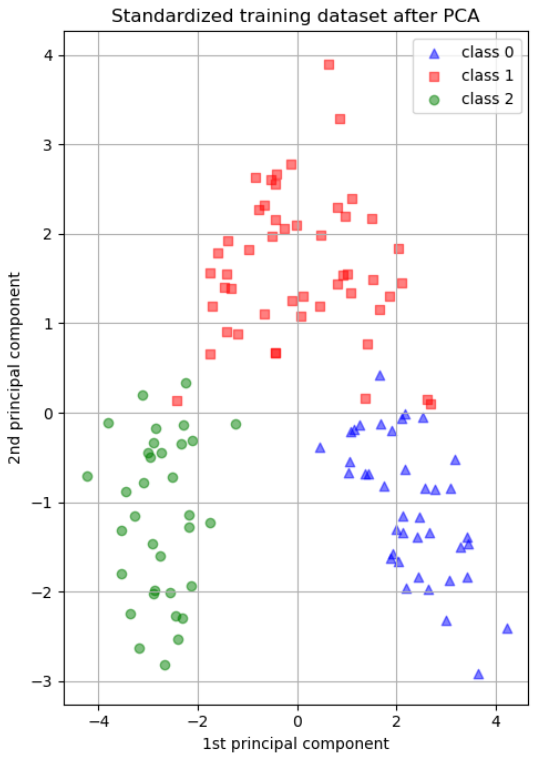


Preprocessing ?

Standardization

- Standardization





Preprocessing for missing value

```
import pandas as pd

data=pd.read_csv('M4_1.csv')
data
```

	A	B	C	D
0	1.0	3	5.0	8
1	NaN	2	5.0	2
2	4.0	7	NaN	2
3	4.0	2	5.0	1

dropna()

```
data.dropna()
```

	A	B	C	D
0	1.0	3	5.0	8
3	4.0	2	5.0	1

dropna(axis=1)

```
data.dropna(axis=1)
```

	B	D
0	3	8
1	2	2
2	7	2
3	2	1

- Remember to assign dropped result to a new dataframe
  - ✓ newdata=data.dropna()

Preprocessing for missing value

```
import pandas as pd

data=pd.read_csv('M4_1.csv')
data
```

	A	B	C	D
0	1.0	3	5.0	8
1	NaN	2	5.0	2
2	4.0	7	NaN	2
3	4.0	2	5.0	1

fillna(value)

```
data.fillna(0)
```

	A	B	C	D
0	1.0	3	5.0	8
1	0.0	2	5.0	2
2	4.0	7	0.0	2
3	4.0	2	5.0	1

Interpolate()

```
data.interpolate()
```

	A	B	C	D
0	1.0	3	5.0	8
1	2.5	2	5.0	2
2	4.0	7	5.0	2
3	4.0	2	5.0	1

Preprocessing for missing value

```
import pandas as pd

data=pd.read_csv('M4_1.csv')
data
```

	A	B	C	D
0	1.0	3	5.0	8
1	NaN	2	5.0	2
2	4.0	7	NaN	2
3	4.0	2	5.0	1

fillna(data.mean())

```
data.fillna(data.mean())
```

	A	B	C	D
0	1.0	3	5.0	8
1	3.0	2	5.0	2
2	4.0	7	5.0	2
3	4.0	2	5.0	1

Preprocessing in scikit-learn

Standardization

Standardize features by removing the mean and scaling to unit variance.

```
from sklearn import preprocessing
import numpy as np

xdata=[1, 2, 3, 4]
xscale=preprocessing.scale(xdata)
print(xscale,'mean=',xscale.mean(),'variance', xscale.var())
```

```
[-1.34164079 -0.4472136   0.4472136   1.34164079] mean= 0.0 variance 1.0
```

Preprocessing in scikit-learn

Standardization

The standardization process can be stored in a module which can be fit first, and then applied to other datasets later.

```
from sklearn import preprocessing

xdata=[1, 1, 0, 0]
scaler=preprocessing.StandardScaler()
scaler.fit(xdata)

standx=scaler.transform(xdata)
print(standx,'mean=',standx.mean(),', variance:', standx.var())

ydata=[1, 2, 3, 4]
ystand=scaler.transform(ydata)
print(ystand)
```

```
[ 1.  1. -1. -1.] mean= 0.0 , variance: 1.0
[1. 3. 5. 7.]
```

- We can use *scaler.fit_transform()* as well.
- The fit function is to find parameter for standardization $\frac{1}{\sigma}\varphi\left(\frac{x-u}{\sigma}\right)$
  - $\sigma = 0.5, u = 0.5$

Preprocessing in scikit-learn

Normalization

Scale input vectors individually to unit norm (vector length).

```
from sklearn import preprocessing

xdata=[1, 2, 3, 4]
print(preprocessing.normalize(xdata,norm='max'))
print(preprocessing.normalize(xdata,norm='l1'))
print(preprocessing.normalize(xdata,norm='l2'))
```

```
[[0.25 0.5  0.75 1.  ]]
[[0.1 0.2 0.3 0.4]]
[[0.18257419 0.36514837 0.54772256 0.73029674]]
```

Preprocessing in scikit-learn

MinMaxScaler

Transforms features by scaling each feature to a given range.

```
from sklearn import preprocessing

xdata=[1, 2, 3, 4]
scale_x=preprocessing.minmax_scale(feature_range=(0,2),X=xdata)
print(scale_x)
```

```
[0.         0.66666667 1.33333333 2.        ]
```

Preprocessing in scikit-learn

MinMaxScaler

We can also create an scaling object to do scaling operation.

```
from sklearn import preprocessing

xdata=[1, 2, 3, 4]
minmaxscaler=preprocessing.MinMaxScaler()
scale_x=minmaxscaler.fit_transform(xdata)
print(scale_x)
ydata=[-2, 0, 6, 10]
print(minmaxscaler.transform(ydata))
```

```
[0.         0.33333333 0.66666667 1.        ]
[-1.         -0.33333333  1.66666667  3.        ]
```

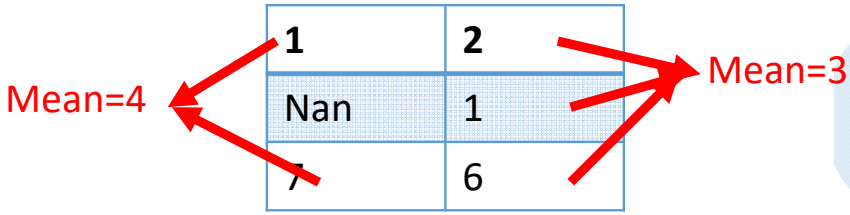
Preprocessing in scikit-learn

- Impute module in preprocessing package can also be used to fill the missing values.

```
import numpy as np
#from sklearn import SimpleImputer
from sklearn.preprocessing import Imputer

imp = Imputer(missing_values=np.nan, strategy='mean', axis=0)
xdata=[[1, 2], [np.nan, 1], [7, 6]]
imp.fit(xdata)
print(imp.transform(xdata))
X = [[np.nan, 2], [9, np.nan], [100, 6],[np.nan, np.nan]]
print(imp.transform(X))
```

Can also specify ***'median'*** or ***'most_frequent'***

```
[[1. 2.]
 [4. 1.]
 [7. 6.]]
[[  4.   2.]
 [  9.   3.]
 [100.   6.]
 [  4.   3.]]
```



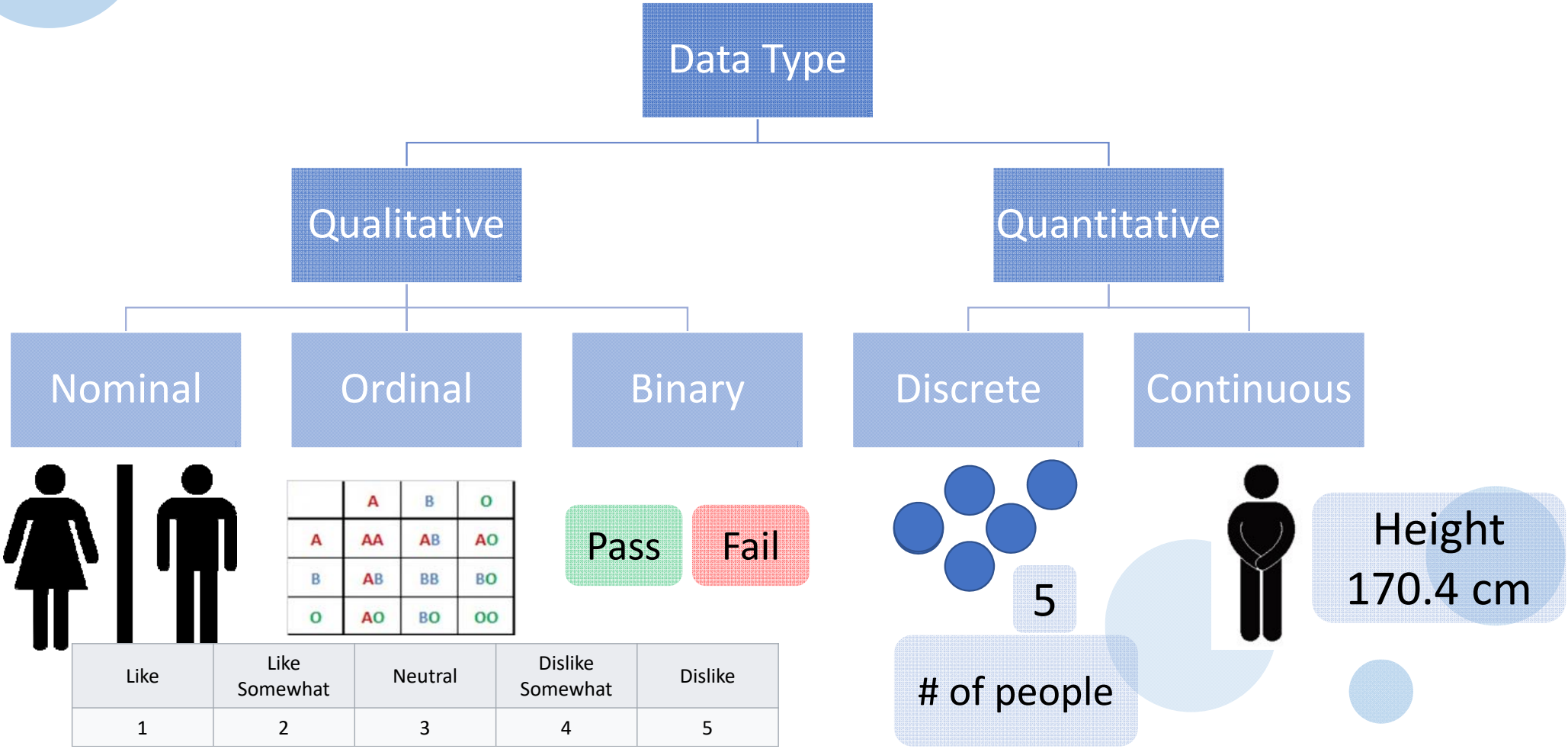
Preprocessing in scikit-learn

- Can also fill the missing values using the most frequent data.

```
import numpy as np
#from sklearn import SimpleImputer
from sklearn.preprocessing import Imputer

imp = Imputer(missing_values=np.nan, strategy='most_frequent', axis=0)
xdata=[[1, 2], [np.nan, 1], [7, 6],[7,2]]
imp.fit(xdata)
print(imp.transform(xdata))
X = [[np.nan, 2], [9, np.nan], [100, 6],[np.nan, np.nan]]
print(imp.transform(X))
```

```
[[1. 2.]
 [7. 1.]
 [7. 6.]
 [7. 2.]]
[[  7.   2.]
 [  9.   2.]
 [100.   6.]
 [  7.   2.]]
```

Type of input variables/feature



Like	Like Somewhat	Neutral	Dislike Somewhat	Dislike
1	2	3	4	5

Encoding of variables/features

Label encoding

Fruit	Label
Apple	1
Orange	2
Banana	3
Grape	4

One-hot encoding

Apple	Orange	Banana	Grape
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Python package used for coding

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(['Apple','Orange','Banana','Grape'])
          0       3       1       2

print(le.transform(['Orange','Grape','Banana','Grape','Apple']))

print(le.inverse_transform([0, 2, 1, 0]))
```

Import package

Encoder

Fit label

Encode data

```
[3 2 1 2 0]
['Apple' 'Grape' 'Banana' 'Apple']
```

One-hot encoding example

```
from sklearn import preprocessing
lo = preprocessing.OneHotEncoder()
le = preprocessing.LabelEncoder()

le_result=le.fit_transform(['Apple','Orange','Banana','Grape'])
lo_result=lo.fit_transform(le_result.reshape(-1,1)).toarray()

print(lo_result)
```

```
[[1. 0. 0. 0.]
 [0. 0. 0. 1.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]]
```

One-hot coding using pandas

- Pandas provides *get_dummies()* function can also provide similar one-hot encoding result.

```
import pandas as pd

data = pd.DataFrame({"顏色" : ["紅色", "紅色", "藍色", "黃色", "黃色"]
                     ,"Item":[10,21,30,41,56]})
data
```

	Item	顏色
0	10	紅色
1	21	紅色
2	30	藍色
3	41	黃色
4	56	黃色

```
pd.get_dummies(data['顏色'])
```

	紅色	藍色	黃色
0	1	0	0
1	1	0	0
2	0	1	0
3	0	0	1
4	0	0	1

```
np.array(pd.get_dummies(data['顏色']))
```

```
array([[1, 0, 0],
       [1, 0, 0],
       [0, 1, 0],
       [0, 0, 1],
       [0, 0, 1]], dtype=uint8)
```

```
pd.get_dummies(data)
```

	Item	顏色_紅色	顏色_藍色	顏色_黃色
0	10	1	0	0
1	21	1	0	0
2	30	0	1	0
3	41	0	0	1
4	56	0	0	1

特徵選取

- 使用pandas 套件找出相關係數，seaborn套件視覺化顯示相關係數的矩陣，範例如下：

```
import seaborn as sns
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True)
```

df.corr() : 產生相關係數的矩陣

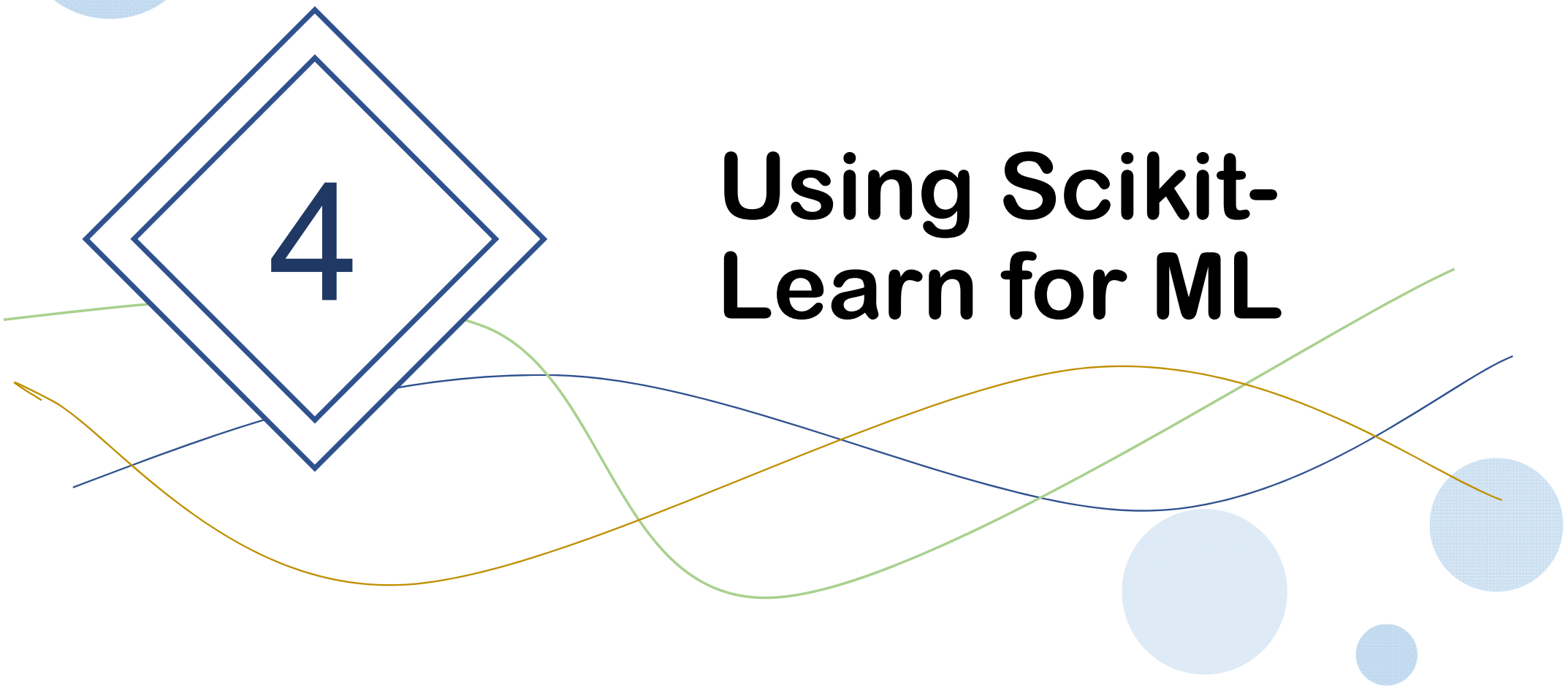
相關係數

- 相關係數為兩個變數之間的共變異數和標準差的商
- 可以被轉換成下面幾種等價的形式

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

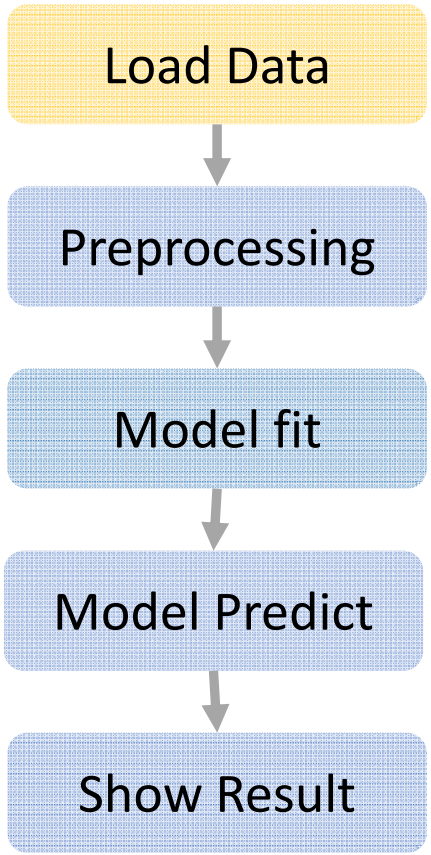
$$r = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \overline{Y})^2}}$$

$$r = \frac{1}{n-1}\sum_{i=1}^{n}\left(\frac{X_i - \overline{X}}{\sigma_X}\right)\left(\frac{Y_i - \overline{Y}}{\sigma_Y}\right)$$



Using Scikit-Learn for ML

Data Loading



- sklearn provides a dataset package which contains some practice data which can be loaded.

```
from sklean import datasets datasets
diabetes = datasets.load_diabetes()
```

- Very often, the data are stored in csv files

```
import pandas as pd
df = pd.read_csv('file.csv',header=None,sep=' ')
```

Sklearn dataset

https://scikit-learn.org/stable/datasets/index.html

6.2. Toy datasets

scikit-learn comes with a few small standard datasets that do not require to download any file from some external website.

They can be loaded using the following functions:

`load_boston` ([return_X_y])	Load and return the boston house-prices dataset (regression).
`load_iris` ([return_X_y])	Load and return the iris dataset (classification).
`load_diabetes` ([return_X_y])	Load and return the diabetes dataset (regression).
`load_digits` ([n_class, return_X_y])	Load and return the digits dataset (classification).
`load_linnerud` ([return_X_y])	Load and return the linnerud dataset (multivariate regression).
`load_wine` ([return_X_y])	Load and return the wine dataset (classification).
`load_breast_cancer` ([return_X_y])	Load and return the breast cancer wisconsin dataset (classification).

csv file of data

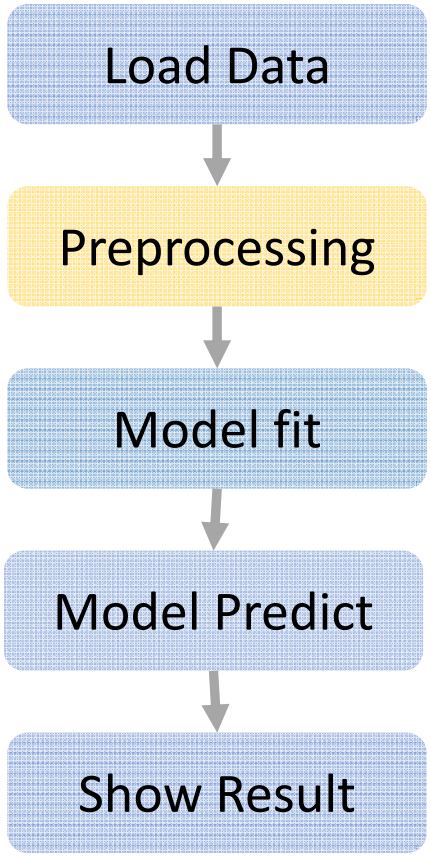
- You have organize your data in the following format

features

cases

ID	gender	date_lt	hcc	preOP_NI	postOP_N	date_exp	LTdate	risk	BIR	age
101	1	5-Jan-05	0	2.9	7.2		16441	3	-2452	51.73
201	1	19-Jan-05	1	1	3		16455	1	-2361	51.52
111	1	25-Jan-05	0	2.8	3.4		16461	1	-1207	48.37
50	1	23-Feb-05	1	2.4	2	9/14/2008	16490	1	-6436	62.77
53	1	2-Mar-05	0	1.9	7.4		16497	3	-2220	51.24
13	1	9-Mar-05	0	2.3	1.8		16504	1	-6930	64.16
92	1	16-Mar-05	1	92+V11:A	4.5	12/5/2005	16511	1	-5294	59.7

Data preparation/preprocessing



- Split dataset into training & testing sets
- Split each set into target and features

```
x_train = x[:-20]
x_test = x[-20:]
y_train = diabetes.target[:-20]
y_test = diabetes.target[-20:]
```

- Can use normalize or other methods

```
from sklearn import preprocessing
x = preprocessing.scale(x)
y = preprocessing.scale(y)
```

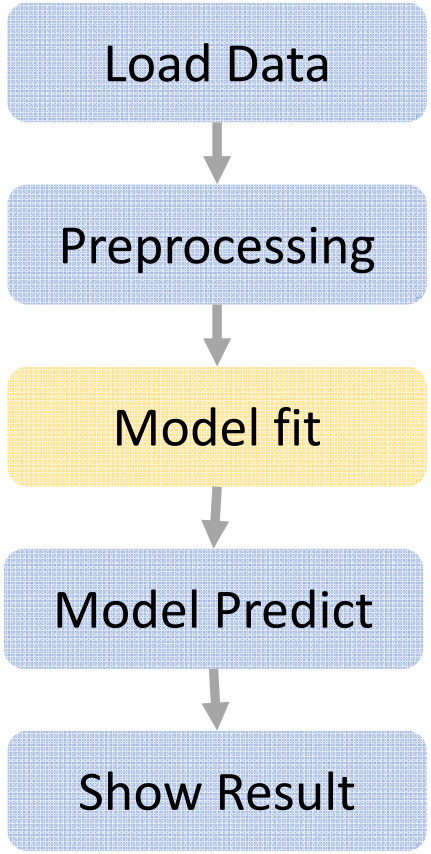
Some quick notes

- The input features **X** should be in the form of 2D array no matter the number of features being used.
  - Ex: X=[[0.5],[1.2],[3.4].....] instead of [0.5, 1.2, 3.4]
- The function ***train_test_split()*** can be used to split the data into the training and testing datasets.

```
import numpy as np
from sklearn.model_selection import train_test_split
X, y = np.arange(10).reshape((5, 2)), range(5)
print('Input X:',X)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42)
print('Training X:',X_train)
print('Training Y:',y_train)
print('Test X:',X_test)
print('Test Y:',y_test)
```

```
Input X: [[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]
Training X: [[4 5]
 [0 1]
 [6 7]]
Training Y: [2, 0, 3]
Test X: [[2 3]
 [8 9]]
Test Y: [1, 4]
```

Model object creation and fit



- Create a model object using the module provided by sklearn packages

from sklearn import linear_model
regr = linear_model.LinearRegression()

- We can set the relevant hyper-parameters

- Feed the data into the model object to fit/train.

regr.fit(x_train, y_train)

Linear model provided by sklearn

`sklearn.linear_model` : Generalized Linear Models

The `sklearn.linear_model` module implements generalized linear models. It includes Ridge regression, Bayesian Regression, Lasso and Elastic Net estimators computed with Least Angle Regression and coordinate descent. It also implements Stochastic Gradient Descent related algorithms.

User guide: See the Generalized Linear Models section for further details.

`linear_model.ARDRegression` ([n_iter, tol, ...])	Bayesian ARD regression.
`linear_model.BayesianRidge` ([n_iter, tol, ...])	Bayesian ridge regression.
`linear_model.ElasticNet` ([alpha, l1_ratio, ...])	Linear regression with combined L1 and L2 priors as regularizer.
`linear_model.ElasticNetCV` ([l1_ratio, eps, ...])	Elastic Net model with iterative fitting along a regularization path.
`linear_model.HuberRegressor` ([epsilon, ...])	Linear regression model that is robust to outliers.
`linear_model.Lars` ([fit_intercept, verbose, ...])	Least Angle Regression model a.k.a.
`linear_model.LarsCV` ([fit_intercept, ...])	Cross-validated Least Angle Regression model.
`linear_model.Lasso` ([alpha, fit_intercept, ...])	Linear Model trained with L1 prior as regularizer (aka the Lasso)
`linear_model.LassoCV` ([eps, n_alphas, ...])	Lasso linear model with iterative fitting along a regularization path.
`linear_model.LassoLars` ([alpha, ...])	Lasso model fit with Least Angle Regression a.k.a.
`linear_model.LassoLarsCV` ([fit_intercept, ...])	Cross-validated Lasso, using the LARS algorithm.
`linear_model.LassoLarsIC` ([criterion, ...])	Lasso model fit with Lars using BIC or AIC for model selection
`linear_model.LinearRegression` ([...])	Ordinary least squares Linear Regression.
`linear_model.LogisticRegression` ([penalty, ...])	Logistic Regression (aka logit, MaxEnt) classifier.
`linear_model.LogisticRegressionCV` ([Cs, ...])	Logistic Regression CV (aka logit, MaxEnt) classifier.
`linear_model.MultiTaskLasso` ([alpha, ...])	Multi-task Lasso model trained with L1/L2 mixed-norm as regularizer.
`linear_model.MultiTaskElasticNet` ([alpha, ...])	Multi-task ElasticNet model trained with L1/L2 mixed-norm as regularizer
`linear_model.MultiTaskLassoCV` ([eps, ...])	Multi-task Lasso model trained with L1/L2 mixed-norm as regularizer.
`linear_model.MultiTaskElasticNetCV` ([...])	Multi-task L1/L2 ElasticNet with built-in cross-validation.
`linear_model.OrthogonalMatchingPursuit` ([...])	Orthogonal Matching Pursuit model (OMP)
`linear_model.OrthogonalMatchingPursuitCV` ([...])	Cross-validated Orthogonal Matching Pursuit model (OMP).
`linear_model.PassiveAggressiveClassifier` ([...])	Passive Aggressive Classifier
`linear_model.PassiveAggressiveRegressor` ([C, ...])	Passive Aggressive Regressor
`linear_model.Perceptron` ([penalty, alpha, ...])	Read more in the User Guide.
`linear_model.RANSACRegressor` ([...])	RANSAC (RANdom SAmple Consensus) algorithm.
`linear_model.Ridge` ([alpha, fit_intercept, ...])	Linear least squares with l2 regularization.
`linear_model.RidgeClassifier` ([alpha, ...])	Classifier using Ridge regression.

Previous | Next | Up

scikit-learn v0.21.2
Other versions

Please cite us if you use the software.

`sklearn.linear_model.LinearRegression`
Examples using
`sklearn.linear_model.LinearRegr`

`sklearn.linear_model.LinearRegression`

class `sklearn.linear_model.` **`LinearRegression`** (*fit_intercept=True, normalize=False, copy_X=True, n_jobs=None*) [source]

Ordinary least squares Linear Regression.

Parameters:

fit_intercept : ***boolean, optional, default True***

whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (e.g. data is expected to be already centered).

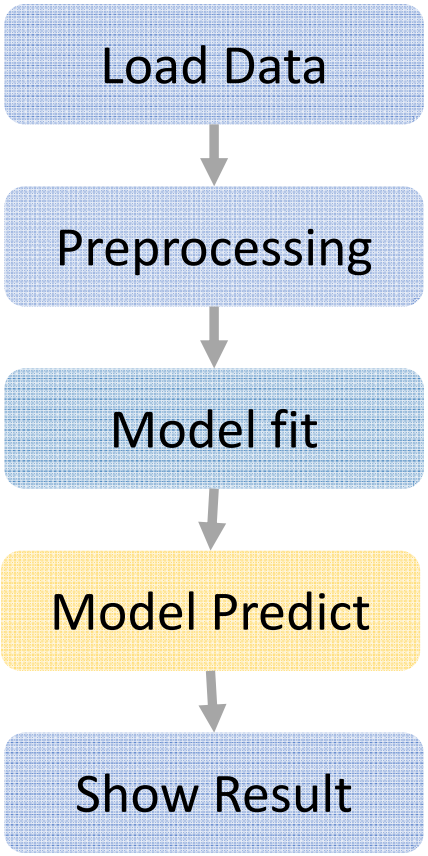
normalize : ***boolean, optional, default False***

This parameter is ignored when `fit_intercept` is set to False. If True, the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm. If you wish to standardize, please use `sklearn.preprocessing.StandardScaler` before calling `fit` on an estimator with `normalize=False`.

Methods

`fit` (self, X, y[, sample_weight])	Fit linear model.
`get_params` (self[, deep])	Get parameters for this estimator.
`predict` (self, X)	Predict using the linear model
`score` (self, X, y[, sample_weight])	Returns the coefficient of determination R^2 of the prediction.
`set_params` (self, **params)	Set the parameters of this estimator.

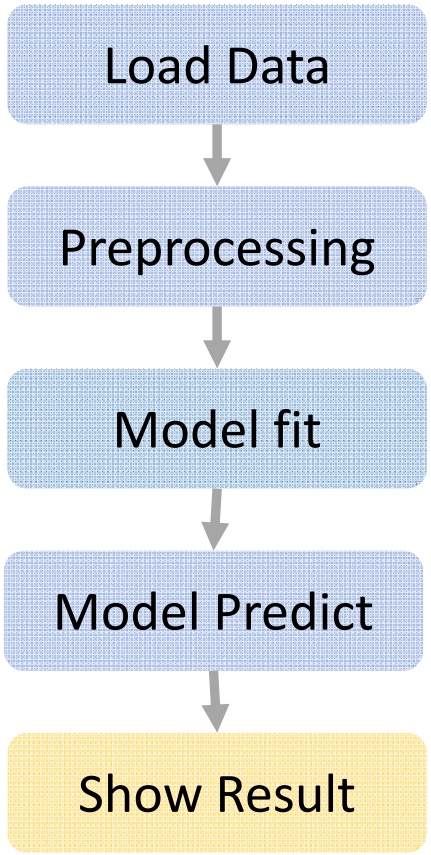
Predict test data using the trained model



- Call the ***predict()*** method of the trained model object.

y_pred = regr.predict(x_test)

Show the predicted result



- Can use graphics to show results

```
import matplotlib.pyplot as plt
plt.scatter(x_test, y_test,  color='blue')
plt.plot(x_test, y_pred, color='red')
plt.show()
```

- Can also use some metrics
  - Can call ***score()*** method of the model object
  - Or use metric module

Metric/score for evaulation

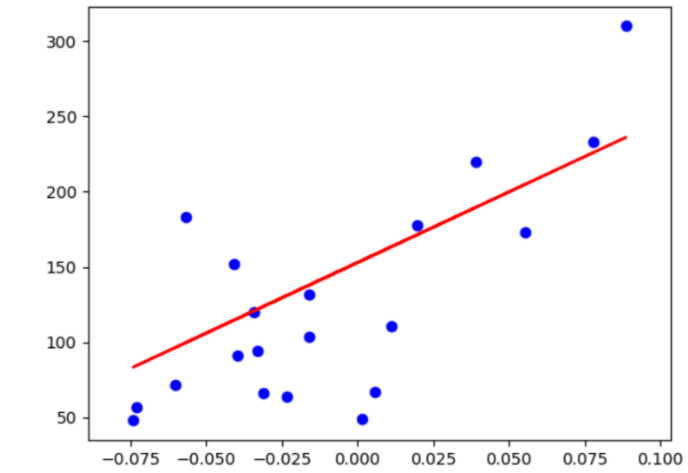
- ***From sklearn import metrics***

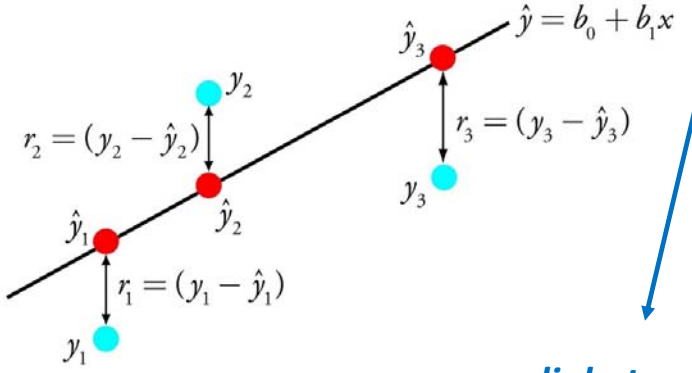
Regression metrics

See the Regression metrics section of the user guide for further details.

`metrics.explained_variance_score` (y_true, y_pred)	Explained variance regression score function
`metrics.max_error` (y_true, y_pred)	max_error metric calculates the maximum residual error.
`metrics.mean_absolute_error` (y_true, y_pred)	Mean absolute error regression loss
`metrics.mean_squared_error` (y_true, y_pred[, …])	Mean squared error regression loss
`metrics.mean_squared_log_error` (y_true, y_pred)	Mean squared logarithmic error regression loss
`metrics.median_absolute_error` (y_true, y_pred)	Median absolute error regression loss
`metrics.r2_score` (y_true, y_pred[, …])	R^2 (coefficient of determination) regression score function.

Overall Linear Regression Example






```python
from sklearn import linear_model, datasets
import numpy as np
import matplotlib.pyplot as plt

diabetes = datasets.load_diabetes()

x = diabetes.data[:, np.newaxis, 2]

x_train = x[:-20]
x_test = x[-20:]

y_train = diabetes.target[:-20]
y_test = diabetes.target[-20:]

regr = linear_model.LinearRegression()
regr.fit(x_train, y_train)
y_pred = regr.predict(x_test)

plt.scatter(x_test, y_test, color='blue')
plt.plot(x_test, y_pred, color='red')

plt.show()
```

Import package

Load dataset

Split dataset

Select model

Fit model

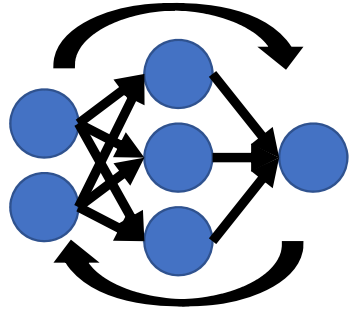
inference

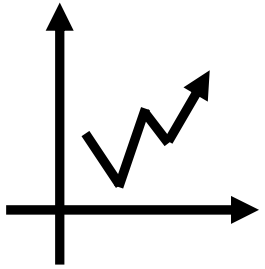
Display result

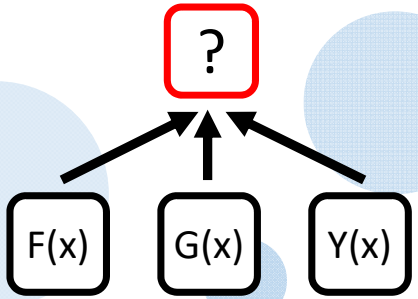
x=diabetes.data[:,2:3]

Model Selection

Training set	Training model
Testing set	Assess the performance
Validation set	Use for model selection

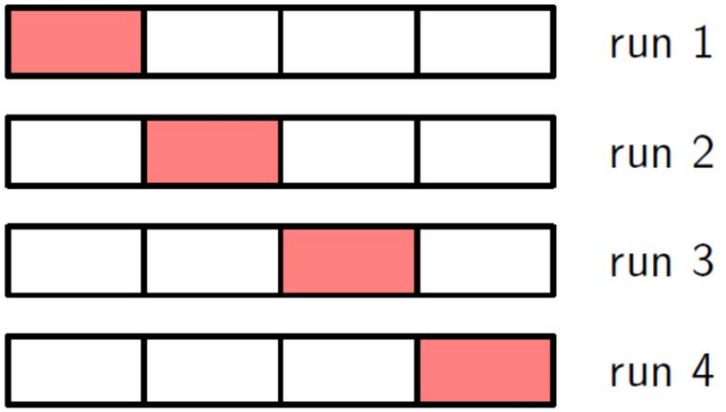




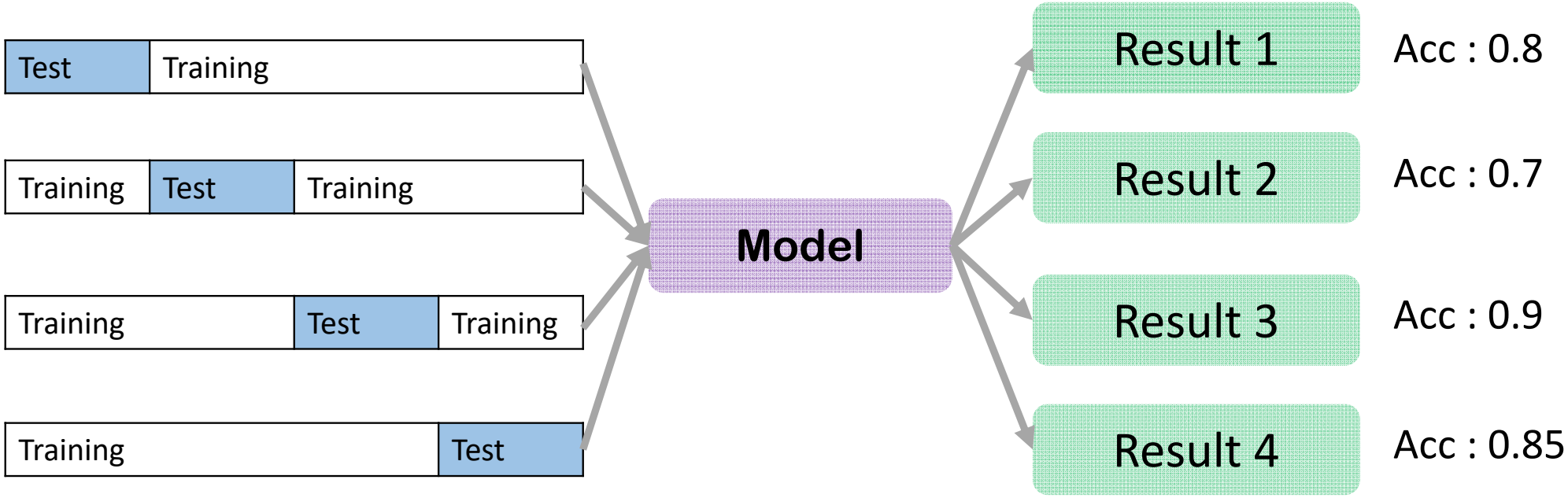


Model Selection

- ***Cross-validation*** method can be applied for limited data, which allows a proportion ***(S –1)/S*** of the available data to be used for training.



Cross-validation



Mean Acc : (0.8 + 0.7 + 0.9 + 0.85)/4 = 0.8125

Cross-validation

```
# from sklearn import datasets, linear_model
from sklearn.model_selection import cross_val_score
diabetes = datasets.load_diabetes()
X = diabetes.data[:150]
y = diabetes.target[:150]
lreg = linear_model.LinearRegression()
print(cross_val_score(lreg, X, y, cv=5))
```

```
[0.36324841 0.28239194 0.4211776  0.30071196 0.61240533]
```