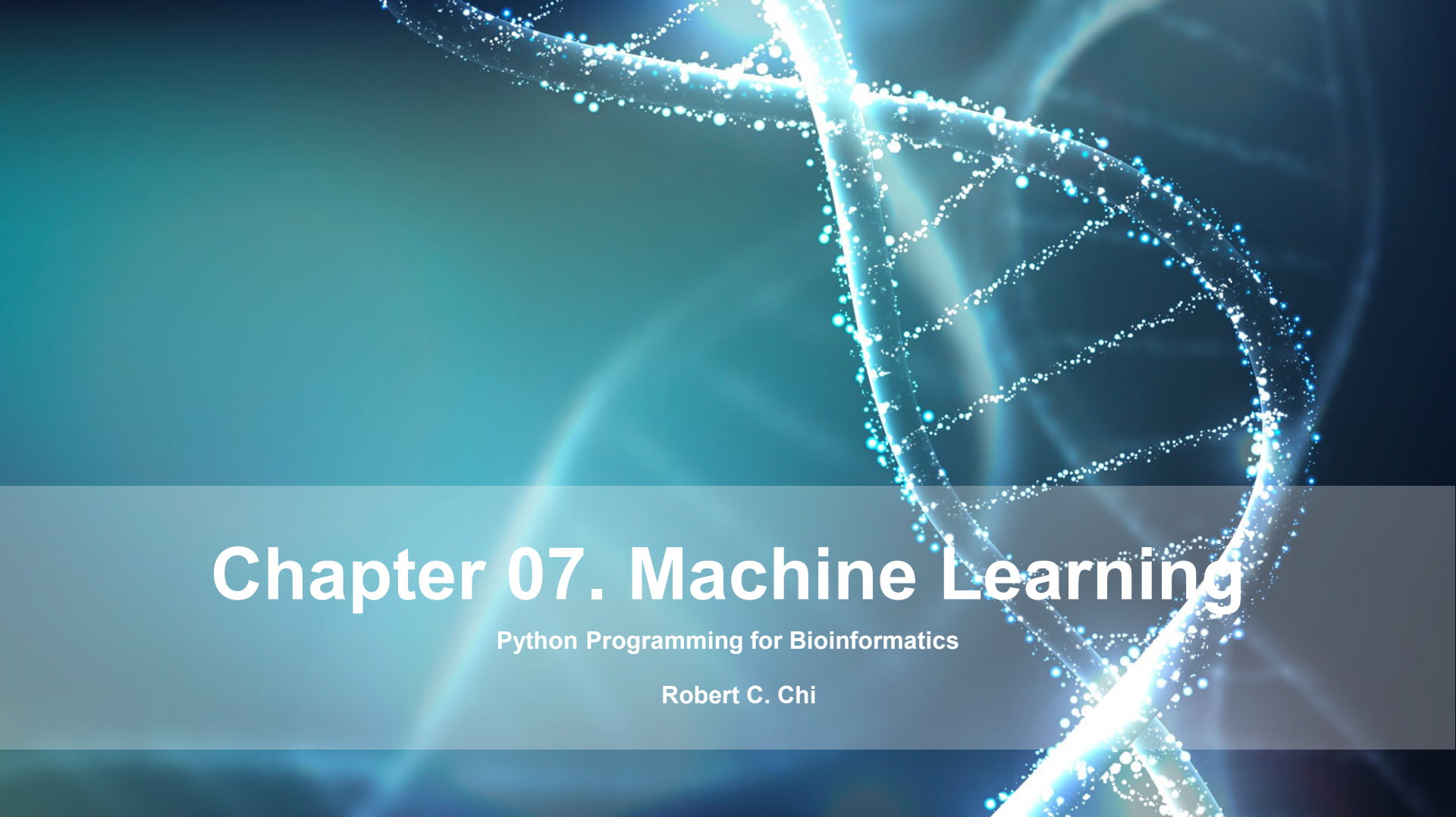


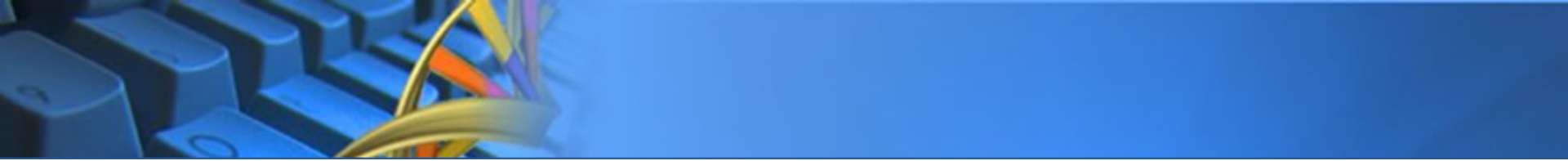
Chapter 07. Machine Learning

Python Programming for Bioinformatics

Robert C. Chi

Agenda

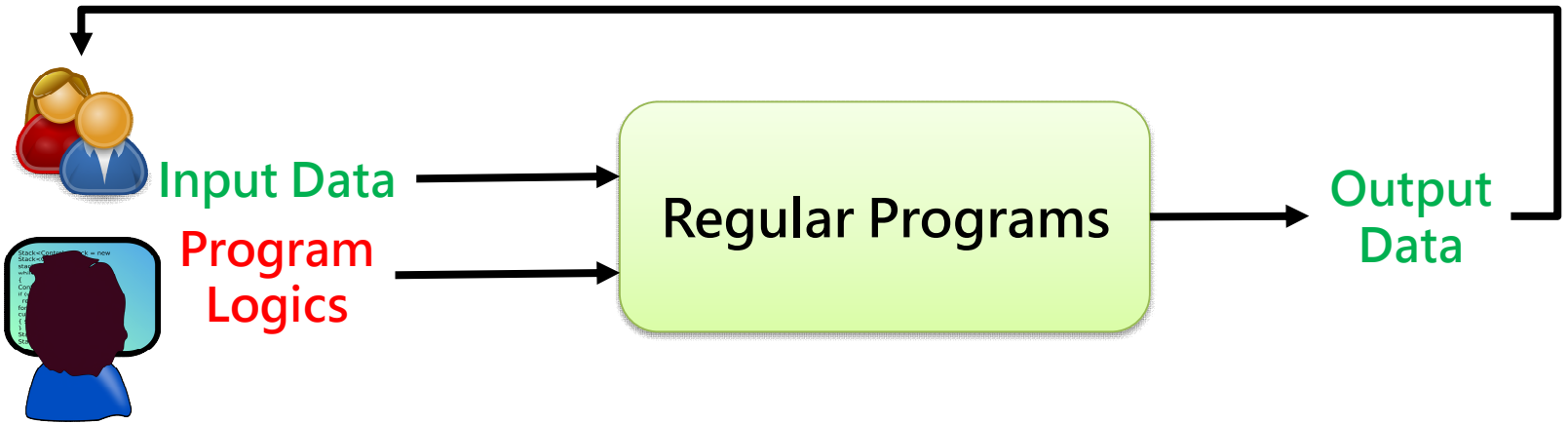
- **Introduction**
- **The Flow of Machine Learning Programs**
- **Classification**
  - Logistic Regression
  - Naive Bayes Classifier
- **Clustering**
  - k-Means Clustering
- **Summary**



INTRODUCTION

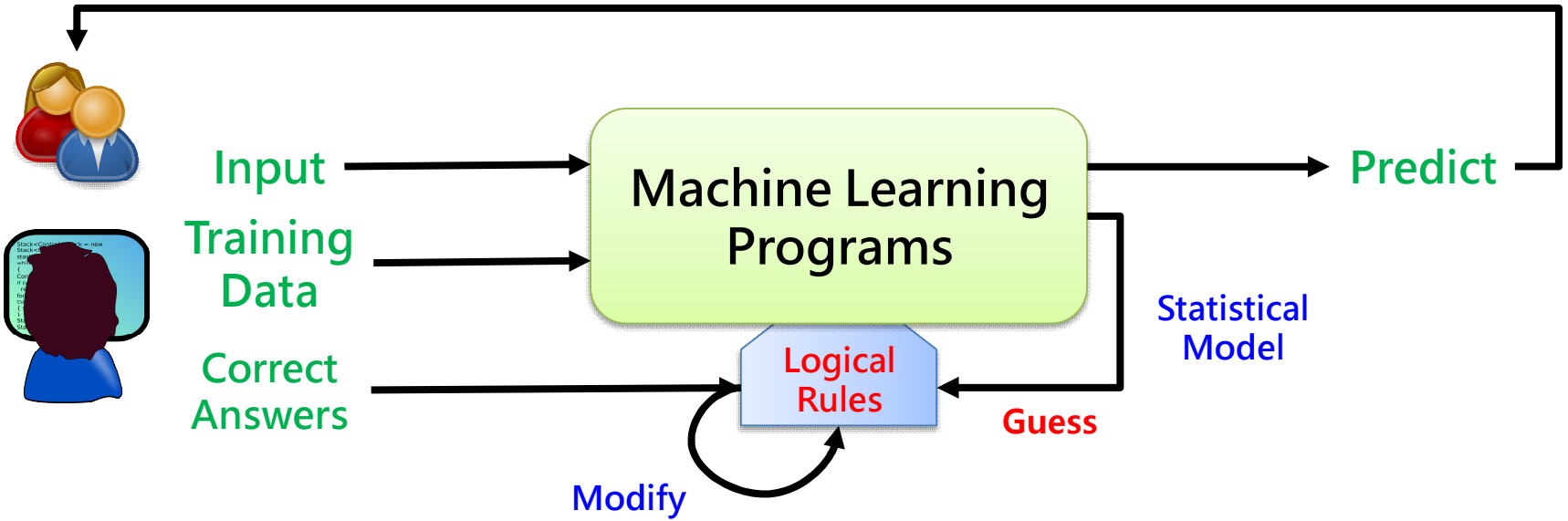
What is Machine Learning Program?

- **Regular Programs**



What is Machine Learning Program?

- **Machine Learning Programs**

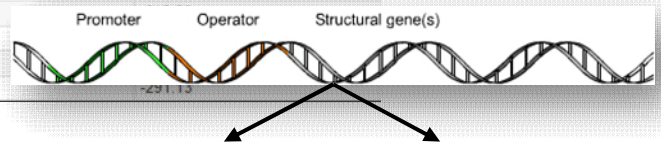


What Machine Learning Can Do?

- **Classification**

Gene pair	Intergene distance	Gene expression score
cotJA-cotJB	-53	-200.78
yesK-yesL	117	-267.14
lplA-lplB	57	-163.47
lplB-lplC	16	-190.30
lplC-lplD	11	-220.94
lplD-yetF	85	-193.94
yfmT-yfmS	16	-182.71
yfmF-yfmE	15	-180.41
citS-citT	-26	-181.73
citM-yflN	58	-259.87
yfiI-yfiJ	126	-414.53
lipB-yfiQ	191	-249.57
yfiU-yfiV	113	-265.28
yfhH-yfhI	145	
cotY-cotX	154	
yjoB-rapA	147	
ptsI-splA	93	

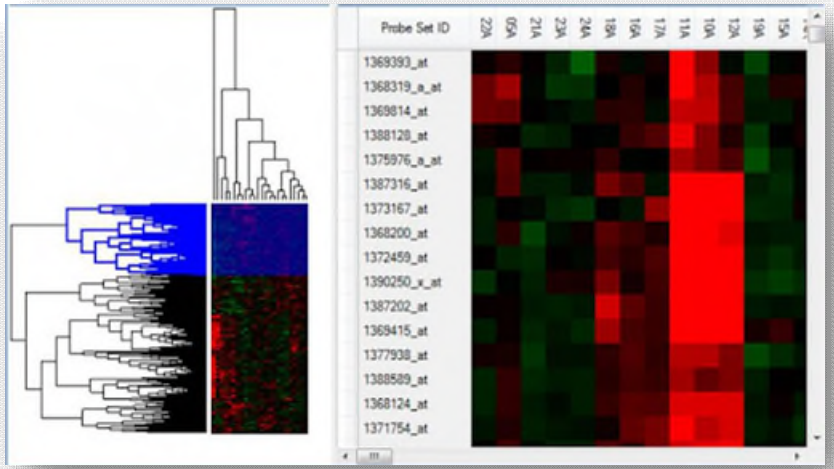
Same Operon?



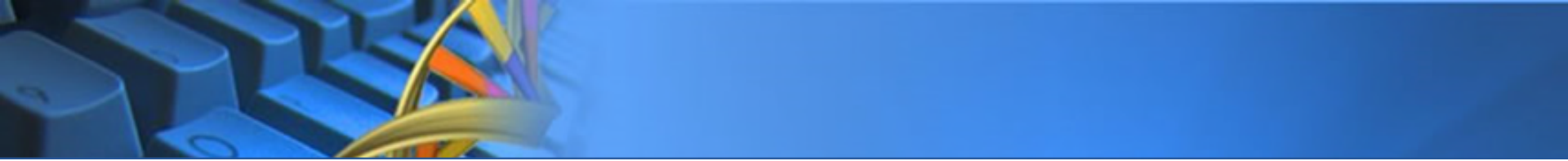
Yes No

- **Clustering**

Microarray Gene Expression

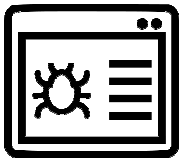


→ Which genes are expressed together?

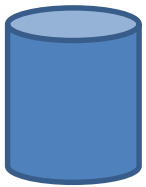


THE FLOW OF MACHINE LEARNING PROGRAMS

Step 1: Collect Data



Web Crawlers



Databases

Feature1 Feature2 Feature3 Feature4

Feature Vector1
Feature Vector2
Feature Vector3
Feature Vector4

	A	B	C	D
1	Country	Age	Salary	ToBuy
2	France	44	72000	No
3	Spain	27	48000	Yes
4	Germany	30	54000	No
5	Spain	38	61000	No
6	Germany	40		Yes
7	France	35	58000	Yes
8	Spain		52000	No
9	France	48	79000	Yes
10	Germany	50	83000	No
11	France	37	67000	Yes

Feature Matrix

Independent Variables

Dependent Variables

ToBuy = a · (Country) + b · (Age) + c · (Salary) a, b, c = Weights

Step 2: Missing Data Makeup

	A	B	C	D
1	Country	Age	Salary	ToBuy
2	France	44	72000	No
3	Spain	27	48000	Yes
4	Germany	30	54000	No
5	Spain	38	61000	No
6	Germany	40	63777.78	Yes
7	France	35	58000	Yes
8	Spain	38.78	52000	No
9	France	48	79000	Yes
10	Germany	50	83000	No
11	France	37	67000	Yes

Strategies

- **Delete**
  - Used When Dataset is Very Large.
- **Average**
  - Take the Average of That Feature.
  - Good for Real Numbers
- **Median**
  - Take the Median of That Feature.
  - Good for Integers
- **Mode (The Most Frequent Value)**
  - Take the Most Frequently Value of That Feature.
  - Good for Categorical Data.

Step 3: Digitalize Categorical Data

- "Category Data" cannot be calculated → Digitalize

	A	B	C	D
1	Country	Age	Salary	ToBuy
2	France	44	72000	No
3	Spain	27	48000	Yes
4	Germany	30	54000	No
5	Spain	38	61000	No
6	Germany	40		Yes
7	France	35	58000	Yes
8	Spain		52000	No
9	France	48	79000	Yes
10	Germany	50	83000	No
11	France	37	67000	Yes

	A	B	C	D
1	Country	Age	Salary	ToBuy
2	1	44	72000	0
3	2	27	48000	1
4	3	30	54000	0
5	2	38	61000	0
6	3	40	63777.78	1
7	1	35	58000	1
8	2	38.78	52000	0
9	1	48	79000	1
10	3	50	83000	0
11	1	37	67000	1

Step 4: Choose the Algorithm

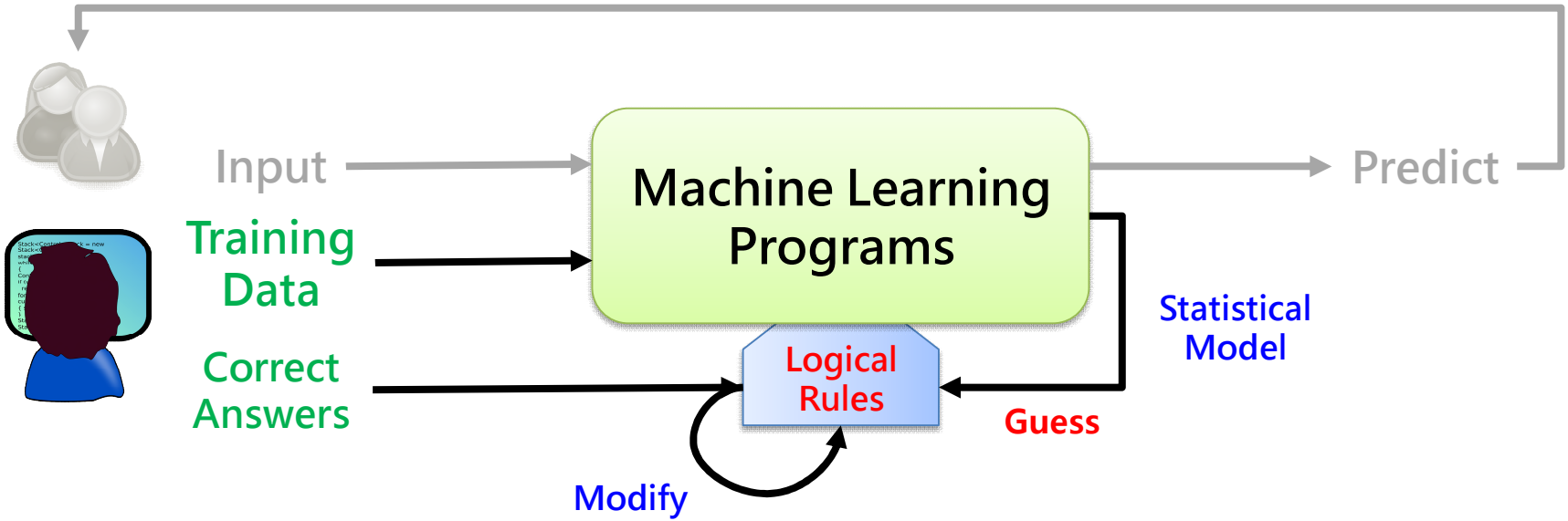
Classification

- k-Nearest Neighbors
- Logistic Regression
- Naïve Bayes Classifier
- Support Vector Machine
- Decision Tree
- …

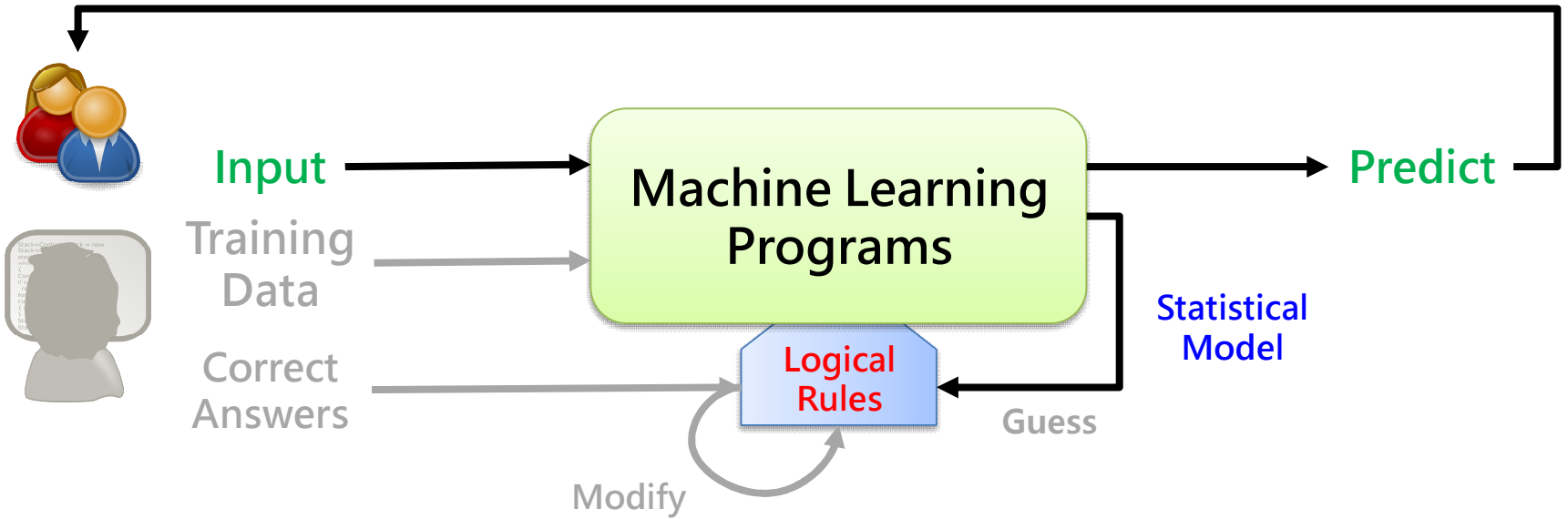
Clustering

- k-Means
- Hierarchical Clustering
- …

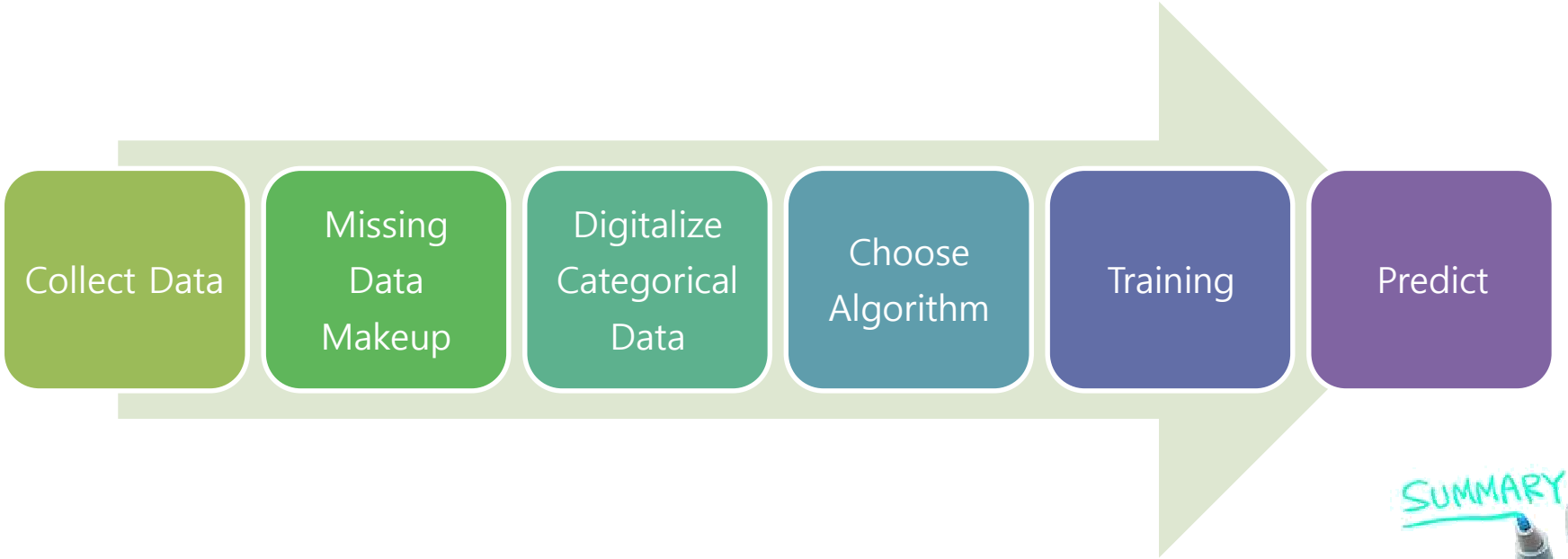
Step 5: Train Your Model

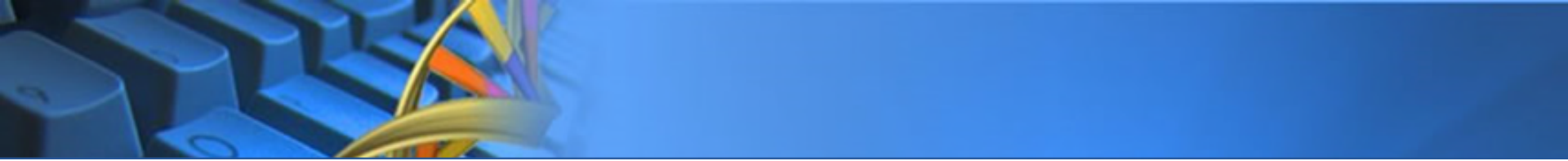


Step 6: Use the Model for Predict



Brief of this Section





Classification

LOGISTIC REGRESSION

How Logistic Regression Works?

Training Data

Gene pair	Intergene distance	Gene expression score	Class
cotJA-cotJB	-53	-200.78	1
yesK-yesL	117	-267.14	1
lplA-lplB	57	-163.47	1
lplB-lplC	16	-190.30	1
lplC-lplD	11	-220.94	1
lplD-yetF	85	-193.94	1
yfmT-yfmS	16	-182.71	1
yfmF-yfmE	15	-180.41	1
citS-citT	-26	-[illegible].73	1
citM-yflN	58	-[illegible]	1
yfiI-yfiJ	126	-414.53	0
lipB-yfiQ	191	-249.57	0
yfiU-yfiV	113	-265.28	0
yfhH-yfhI	145	-312.99	0
cotY-cotX	154	-213.83	0
yjoB-rapA	147	-380.85	0
ptsI-splA	93	-291.13	0

X — Independent Variables

Y — Dependent Variable

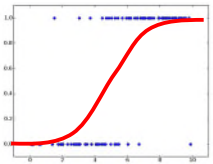
$$y = c_0 + c_1 X_1 \cdots c_n X_n$$

X_1: Inter-gene Distance
X_2: Gene Expression Score
C_i : Parameters to be Estimated
y : At the Same Operon (True/False).

X_i: Continuous Numbers
y: Discrete Numbers

Sigmoid Function

$$p = \frac{1}{1 + e^{-Y}} \rightarrow Y = ln\left(\frac{p}{1-p}\right)$$



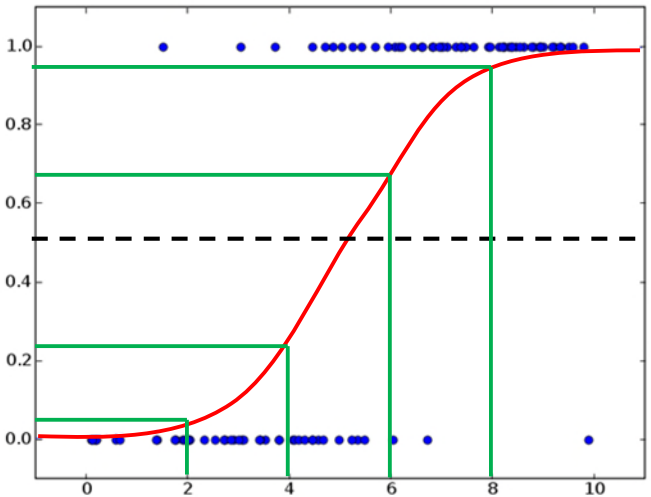
$$ln\left(\frac{p}{1-p}\right) = c_0 + c_1 X_1 \cdots c_n X_n$$

How Logistic Regression Works?

$$ln\left(\frac{p}{1-p}\right) = c_0 + c_1X_1 + c_2X_2$$

$$Y = \begin{cases} 0 & if\ p \leq 0.5 \\ 1 & if\ p > 0.5 \end{cases}$$

Threshold
$P = 0.5$



$X_1 = 2 \rightarrow P = 0.13 \rightarrow Y = 0$

$X_2 = 4 \rightarrow P = 0.22 \rightarrow Y = 0$

$X_3 = 6 \rightarrow P = 0.68 \rightarrow Y = 1$

$X_4 = 8 \rightarrow P = 0.95 \rightarrow Y = 1$

Logistic Regression
- Only Support for Binary Classification.

Pre-processing

```
1  import os
2
3  # Download the CSV File for known Bacillus subtilis operons
4  if not os.path.isfile("Bacillus_subtilis_operons.csv"):
5    os.system("wget -c https://bit.ly/31M0aou -O Bacillus_subtilis_operons.csv")
6
7  # Load Dataset
8  import pandas as pd
9  dataset = pd.read_csv("Bacillus_subtilis_operons.csv")
10
11 # Decompose dataset as X and Y
12 X = dataset.iloc[:, 1:3].values
13 Y = dataset.iloc[:, 3].values
```

dataset =

Gene pair	Intergene distance	Gene expression score	Class
cotJA-cotJB	-53	-200.78	1
yesK-yesL	117	-267.14	1
lplA-lplB	57	-163.47	1
lplB-lplC	16	-190.30	1
lplC-lplD	11	-220.94	1
lplD-yetF	85	-193.94	1
yfmT-yfmS	16	-182.71	1
yfmF-yfmE	15	-180.41	1
citS-citT	-26	-1.73	1
citM-yflN	58	-259.87	1
yfiI-yfiJ	126	-414.53	0
lipB-yfiQ	191	-249.57	0
yfiU-yfiV	113	-265.28	0
yfhH-yfhI	145	-312.99	0
cotY-cotX	154	-213.83	0
yjoB-rapA	147	-380.85	0
ptsI-splA	93	-291.13	0

X

Y

Practice

- **Pre-processing Data for Logistic Regression**
  - Write and Run the following codes on a Colab page called “MachineLearning.ipynb”:

```
1  import os
2
3  # Download the CSV File for known Bacillus subtilis operons
4  if not os.path.isfile("Bacillus_subtilis_operons.csv"):
5    os.system("wget -c https://bit.ly/31M0aou -O Bacillus_subtilis_operons.csv")
6
7  # Load Dataset
8  import pandas as pd
9  dataset = pd.read_csv("Bacillus_subtilis_operons.csv")
10
11 # Decompose dataset as X and Y
12 X = dataset.iloc[:, 1:3].values
13 Y = dataset.iloc[:, 3].values
```

(Solution URL of this Practice)

Training

```
1  # Training
2  from Bio import LogisticRegression
3  model = LogisticRegression.train(X, Y)
4
5  # Show Model Coefficients
6  print(model.beta)
```

[8.98302901571447, -0.03596896044485089, 0.021813956629835193]

$$ln\left(\frac{p}{1-p}\right) = c_0 + c_1X_1 + c_2X_2$$

Predicting

```
1  # Predict User Input Data
2
3  # yxcE-yxcD, X=[6, -173.143442352]
4  print("yxcE-yxcD:", LogisticRegression.classify(model, [6, -173.143442352]))
5
6  # yxiB-yxiA, X=[309, -271.005880394]
7  print("yxiB-yxiA:", LogisticRegression.classify(model, [309, -271.005880394]))
```

```
yxcE-yxcD: 1
yxiB-yxiA: 0
```

Practice

- **Training & Predicting for Logistic Regression**
  - Write and Run the following codes on a Colab page called “MachineLearning.ipynb”:

```
# Training
from Bio import LogisticRegression
model = LogisticRegression.train(X, Y)

# Show Model Coefficients
print(model.beta)

# Predict User Input Data

# yxcE-yxcD, X=[6, -173.143442352]
print("yxcE-yxcD:", LogisticRegression.classify(model, [6, -173.143442352]))

# yxiB-yxiA, X=[309, -271.005880394]
print("yxiB-yxiA:", LogisticRegression.classify(model, [309, -271.005880394]))
```

(Solution URL of this Practice)

Performance Measurement

```
# Show the confidence for predictions

# yxcE-yxcD
q_0, p_1 = LogisticRegression.calculate(model, [6, -173.143442352])
print("Adjacent Gene: yxcE-yxcD")
print("Confidence of Probability for 1: {:.2%}".format(p_1))
print("Confidence of Probability for 0: {:.2%}".format(q_0))

# yxiB-yxiA
q_0, p_1 = LogisticRegression.calculate(model, [309, -271.005880394])
print("Adjacent Gene: yxiB-yxiA")
print("Confidence of Probability for 1: {:.2%}".format(p_1))
print("Confidence of Probability for 0: {:.2%}".format(q_0))
```

```
Adjacent Gene: yxcE-yxcD
Confidence of Probability for 1: 99.32%
Confidence of Probability for 0: 0.68%
```

```
Adjacent Gene: yxiB-yxiA
Confidence of Probability for 1: 0.03%
Confidence of Probability for 0: 99.97%
```

Practice

- **Performance Measurement for Logistic Regression**
  - Write and Run the following codes on a Colab page called “MachineLearning.ipynb”:

```
# Show the confidence for predictions

# yxcE-yxcD
q_0, p_1 = LogisticRegression.calculate(model, [6, -173.143442352])
print("Adjacent Gene: yxcE-yxcD")
print("Confidence of Probability for 1: {:.2%}".format(p_1))
print("Confidence of Probability for 0: {:.2%}".format(q_0))

# yxiB-yxiA
q_0, p_1 = LogisticRegression.calculate(model, [309, -271.005880394])
print("Adjacent Gene: yxiB-yxiA")
print("Confidence of Probability for 1: {:.2%}".format(p_1))
print("Confidence of Probability for 0: {:.2%}".format(q_0))
```

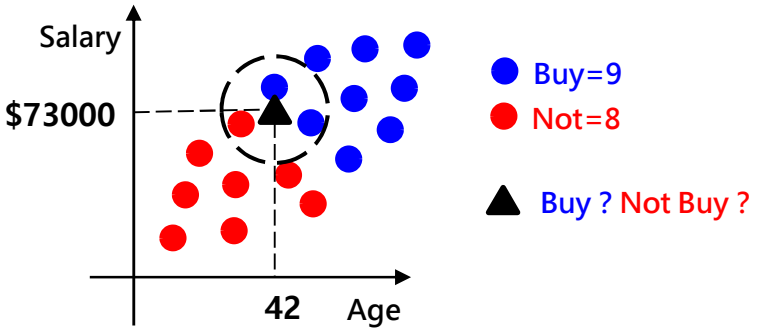
(Solution URL of this Practice)

Classification

NAÏVE BAYES CLASSIFIER

How Naïve Bayes Works?

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$



$P(X = (42, \$73000)) =$ ◯

$=$ Probability Similar to ▲

$= \frac{3}{8+9} = \frac{3}{17}$

$$P(Y = \ ? \ | X = (42, \$73000))$$

$$= \begin{cases} P(Y = ToBuy | X = (42, \$73000)) \\ P(Y = NotBuy | X = (42, \$73000)) \end{cases}$$

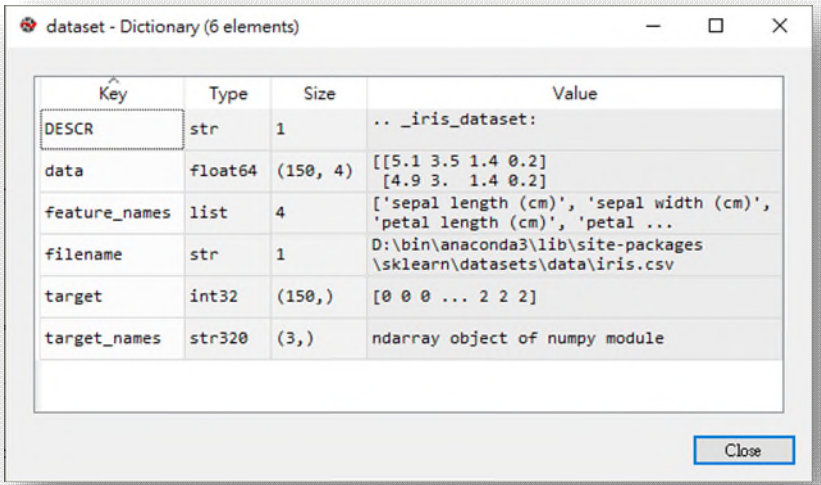
$$= \begin{cases} \dfrac{P(X = (42, \$73000) | Y = ToBuy) P(Y = ToBuy)}{P(X = (42, \$73000))} \\ \dfrac{P(X = (42, \$73000) | Y = NotBuy) P(Y = NotBuy)}{P(X = (42, \$73000))} \end{cases}$$

$$= \begin{cases} \dfrac{2/9 \times 9/17}{3/17} = \dfrac{2}{3} \ (ToBuy) \\ \dfrac{1/8 \times 8/17}{3/17} = \dfrac{1}{3} \ (NotBuy) \end{cases}$$

To Buy ! !

Pre-processing

```
1  from sklearn.datasets import load_iris
2
3  dataset = load_iris()
```



- **DESCR (text)**
  - Description of this Dataset
- **data (array)**
  - Features (Independent Variables X)
- **feature_names (list)**
  - Name of each feature
- **target (array)**
  - Dependent Variable Y
- **target_name (array)**
  - Mapping of Y values and corresponding names
  - e.g., 0=setosa, 1=versicolor… etc.

Pre-processing

```
# Decompose as X and Y
X = dataset.data
Y = dataset.target
Y_types = dataset.target_names.tolist()
```

X

sepal length	sepal width	petal length	petal width
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
...			

Y

Class
0
1
2

Iris Versicolor | Iris Setosa | Iris Virginica

Y_types

[0] [1] [2]

['setosa', 'versicolor', 'virginica']

Practice

- **Pre-processing Data for Naïve Bayes Classifier**
  - Write and Run the following codes on a Colab page called "MachineLearning.ipynb":

```
# Load Dataset
from sklearn.datasets import load_iris

dataset = load_iris()
# Decompose as X and Y
X = dataset.data
Y = dataset.target
Y_types = dataset.target_names.tolist()
```

(Solution URL of this Practice)

Training & Predicting

```
1  from Bio import NaiveBayes
2
3  # Training
4  model = NaiveBayes.train(X, Y)
5
6  # Predict
7  pred1 = NaiveBayes.classify(model, [6.4, 3.8, 6.9, 1.8])
8  print(pred1, Y_types[pred1])
9
10 pred2 = NaiveBayes.classify(model, [5.1, 3.0, 1.3, 0.2])
11 print(pred2, Y_types[pred2])
```

```
2 virginica
0 setosa
```

Practice

- **Training & Predicting for Naïve Bayes Classifier**
  - Write and Run the following codes on a Colab page called "MachineLearning.ipynb":

```
1  from Bio import NaiveBayes
2
3  # Training
4  model = NaiveBayes.train(X, Y)
5
6  # Predict
7  pred1 = NaiveBayes.classify(model, [6.4, 3.8, 6.9, 1.8])
8  print(pred1, Y_types[pred1])
9
10 pred2 = NaiveBayes.classify(model, [5.1, 3.0, 1.3, 0.2])
11 print(pred2, Y_types[pred2])
```

(Solution URL of this Practice)

Performance Measurement

```
1  # Show the confidence for predictions
2  print("Log Likelihood #1:")
3  print(NaiveBayes.calculate(model, [6.4, 3.8, 6.9, 1.8]))
4
5  print("Log Likelihood #2:")
6  print(NaiveBayes.calculate(model, [5.1, 3.0, 1.3, 0.2]))
```

```
Log Likelihood #1:
{0: -2075.950924627617, 1: -1389.780566915392, 2: -12.046223944588279}

Log Likelihood #2:
{0: -7.562297320431017, 1: -1388.394272554272, 2: -2074.852312338949}
```

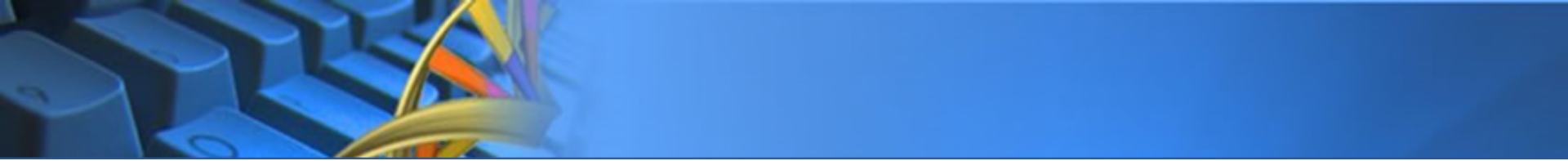
Practice

- **Performance Measurement for Naïve Bayes Classifier**
  - Write and Run the following codes on a Colab page called "MachineLearning.ipynb":

```
# Show the confidence for predictions
print("Log Likelihood #1:")
print(NaiveBayes.calculate(model, [6.4, 3.8, 6.9, 1.8]))

print("Log Likelihood #2:")
print(NaiveBayes.calculate(model, [5.1, 3.0, 1.3, 0.2]))
```

(Solution URL of this Practice)



Clustering

K-MEANS CLUSTERING

What is “Clustering”?

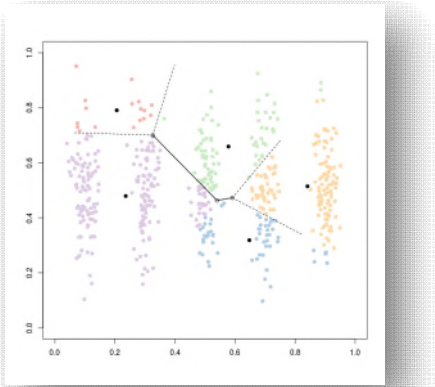
Genre	Age	Income (k$)	Spending
Male	19	15	39
Male	21	15	81
Female	20	16	6
Female	23	16	77
Female	31	17	40
Female	22	17	76
Female	35	18	6
Female	23	18	94
Male	64	19	3

X

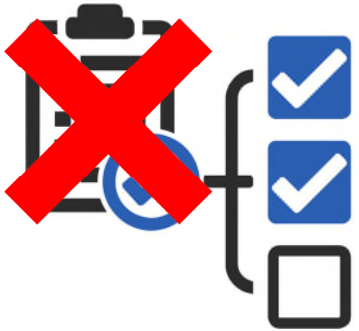
Independent Variables

?

Only has X,
No Y

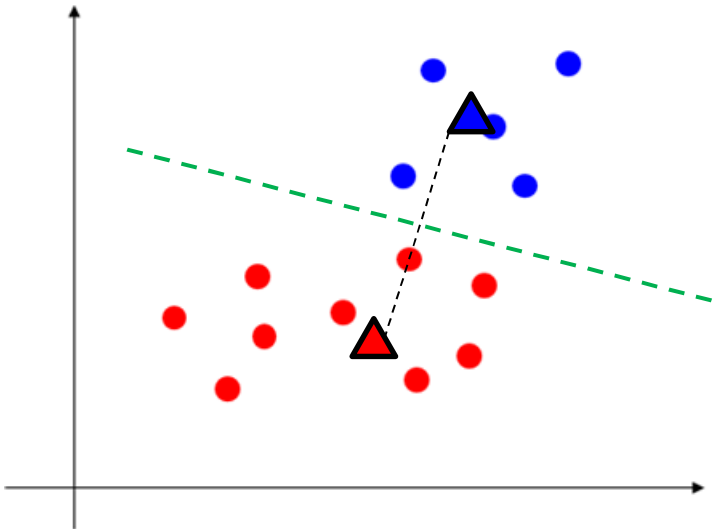


Use Similarity (Distance) to do the job



a.k.a. “Non-Supervised Learning”
(No Correct Answers)

What is K-Means Clustering



1. Suppose to cluster as K groups, choose arbitrary K centroids.
2. Assign each sample to the nearest centroid.
3. Re-calculate all centroids for each group.
4. Repeat (2) ~ (3).
5. If there is no change in group belonging for all samples, the process ends.

K = Optimized by Biopython Automatically

Pre-processing

```
1  # Sequences to be clustered
2  sequence = [ 'AGCT','CGTA','AAGT','TCCG']
3
4  # Digitalized Categorical Data
5  import numpy as np
6
7  matrix = np.asarray([np.fromstring(s, dtype=np.uint8) for s in sequence])
8  print(matrix)
```

```
[[65 71 67 84]
 [67 71 84 65]
 [65 65 71 84]
 [84 67 67 71]]
```

Practice

- **Pre-processing Data for K-Means Clustering**
  - Write and Run the following codes on a Colab page called “MachineLearning.ipynb”:

```
# Sequences to be clustered
sequence = [ 'AGCT','CGTA','AAGT','TCCG']

# Digitalized Categorical Data
import numpy as np

matrix = np.asarray([np.fromstring(s, dtype=np.uint8) for s in sequence])
print(matrix)
```

(Solution URL of this Practice)

Clustering

```
1  from Bio.Cluster import kcluster
2
3  clusterid, error, found = kcluster(matrix)
4  print("Cluster Result for Each Sample:", clusterid)
5  print("Within Cluster Sum of Square (WCSS):", error)
6  print("The number of times the solution was found:", found)
```

```
Cluster Result for Each Sample: [1 0 1 0]
Within Cluster Sum of Square (WCSS): 85.25
The number of times the solution was found: 1
```

Practice

- **Clustering for K-Means**
  - Write and Run the following codes on a Colab page called “MachineLearning.ipynb”:

```
from Bio.Cluster import kcluster

clusterid, error, found = kcluster(matrix)
print("Cluster Result for Each Sample:", clusterid)
print("Within Cluster Sum of Square (WCSS):", error)
print("The number of times the solution was found:", found)
```

(Solution URL of this Practice)

Summary

- **What Machine Learning Can Do?**
  - Classification
  - Clustering
- **Flow of Machine Learning Programs**
  - Collect Data
  - Missing Data Makeup
  - Digitalize Categorical Data
  - Choose the Algorithm
  - Train Your Model
  - Use the Model for Predict
- **Classification**
  - Logistic Regression
  - Naive Bayes Classifier
- **Clustering**
  - K-Means