

Artificial Intelligence

Machine Learning: Supervised and Unsupervised Learning

1111AI05

MBA, IM, NTPU (M6132) (Fall 2022)

Wed 2, 3, 4 (9:10-12:00) (B8F40)



<https://meet.google.com/miy-fbif-max>



Min-Yuh Day, Ph.D,
Associate Professor

[Institute of Information Management, National Taipei University](https://web.ntpu.edu.tw/~myday)

<https://web.ntpu.edu.tw/~myday>



Syllabus

Week	Date	Subject/Topics
1	2022/09/14	Introduction to Artificial Intelligence
2	2022/09/21	Artificial Intelligence and Intelligent Agents
3	2022/09/28	Problem Solving
4	2022/10/05	Knowledge, Reasoning and Knowledge Representation; Uncertain Knowledge and Reasoning
5	2022/10/12	Case Study on Artificial Intelligence I
6	2022/10/19	Machine Learning: Supervised and Unsupervised Learning

Syllabus

Week	Date	Subject/Topics
7	2022/10/26	The Theory of Learning and Ensemble Learning
8	2022/11/02	Midterm Project Report
9	2022/11/09	Deep Learning and Reinforcement Learning
10	2022/11/16	Deep Learning for Natural Language Processing
11	2022/11/23	Invited Talk: AI for Information Retrieval
12	2022/11/30	Case Study on Artificial Intelligence II

Syllabus

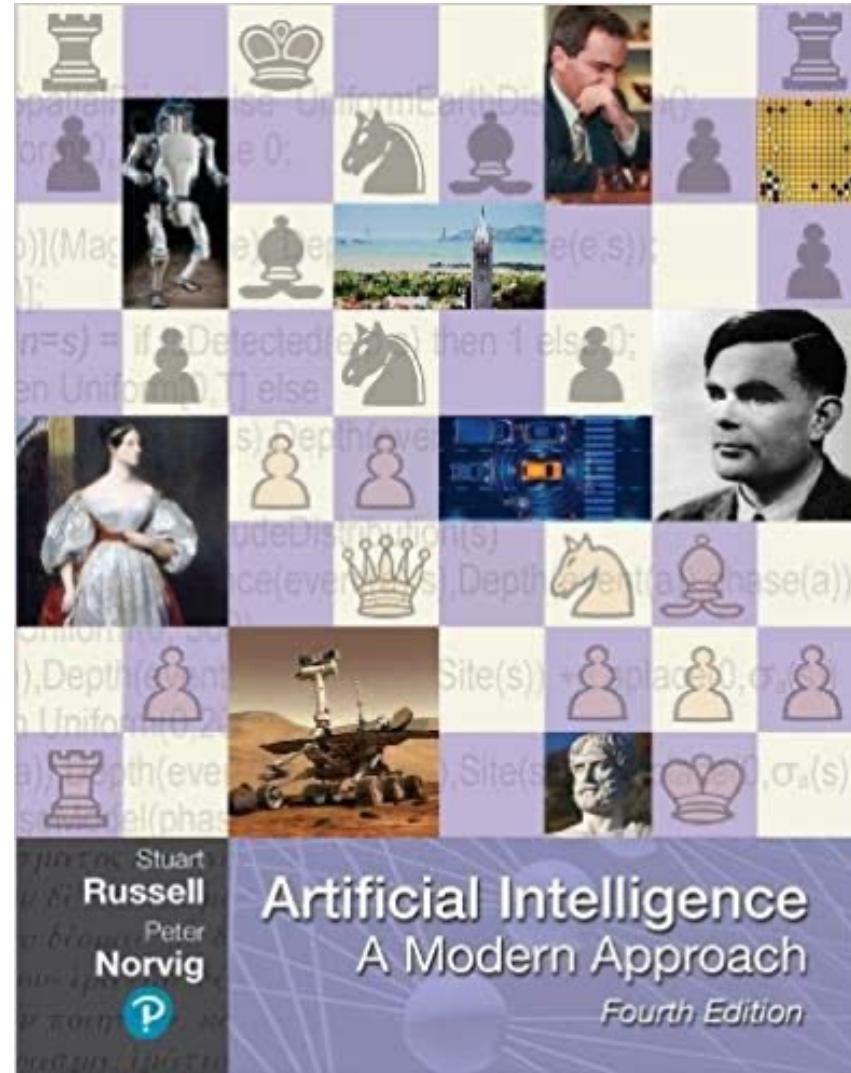
Week	Date	Subject/Topics
13	2022/12/07	Computer Vision and Robotics
14	2022/12/14	Philosophy and Ethics of AI and the Future of AI
15	2022/12/21	Final Project Report I
16	2022/12/28	Final Project Report II
17	2023/01/04	Self-learning
18	2023/01/11	Self-learning

Machine Learning: Supervised and Unsupervised Learning

Outline

- **Machine Learning**
- **Supervised Learning**
- **Unsupervised Learning**

Stuart Russell and Peter Norvig (2020),
Artificial Intelligence: A Modern Approach,
4th Edition, Pearson



Source: Stuart Russell and Peter Norvig (2020), Artificial Intelligence: A Modern Approach, 4th Edition, Pearson

<https://www.amazon.com/Artificial-Intelligence-A-Modern-Approach/dp/0134610997/>

Artificial Intelligence: A Modern Approach

1. Artificial Intelligence
2. Problem Solving
3. Knowledge and Reasoning
4. Uncertain Knowledge and Reasoning
5. Machine Learning
6. Communicating, Perceiving, and Acting
7. Philosophy and Ethics of AI

Artificial Intelligence: Machine Learning

Artificial Intelligence:

5. Machine Learning

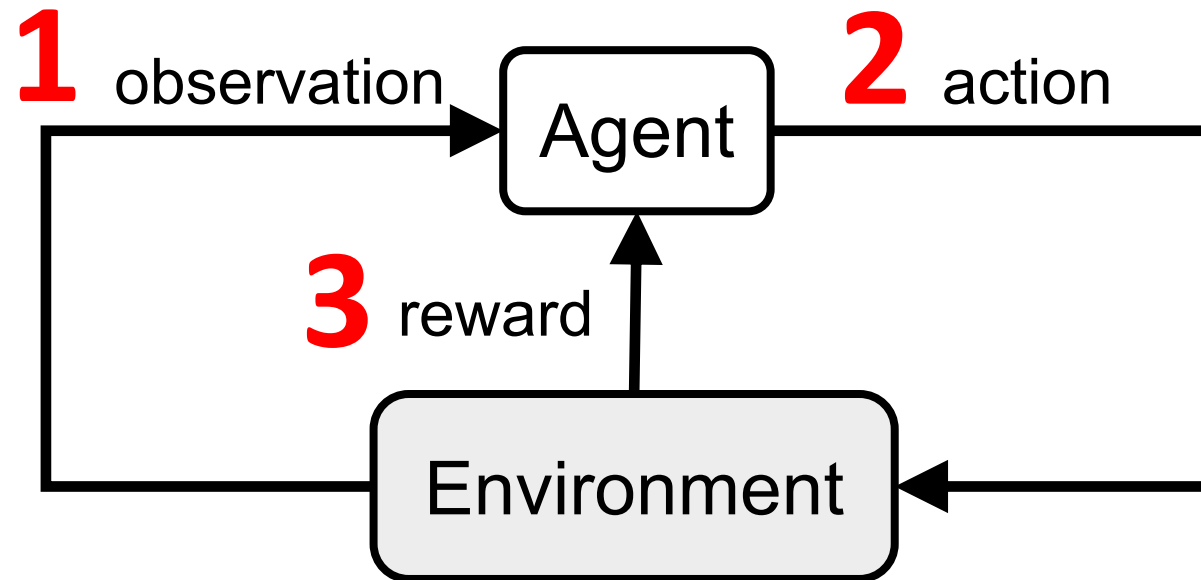
- **Learning from Examples**
- **Learning Probabilistic Models**
- **Deep Learning**
- **Reinforcement Learning**

Reinforcement Learning (DL)

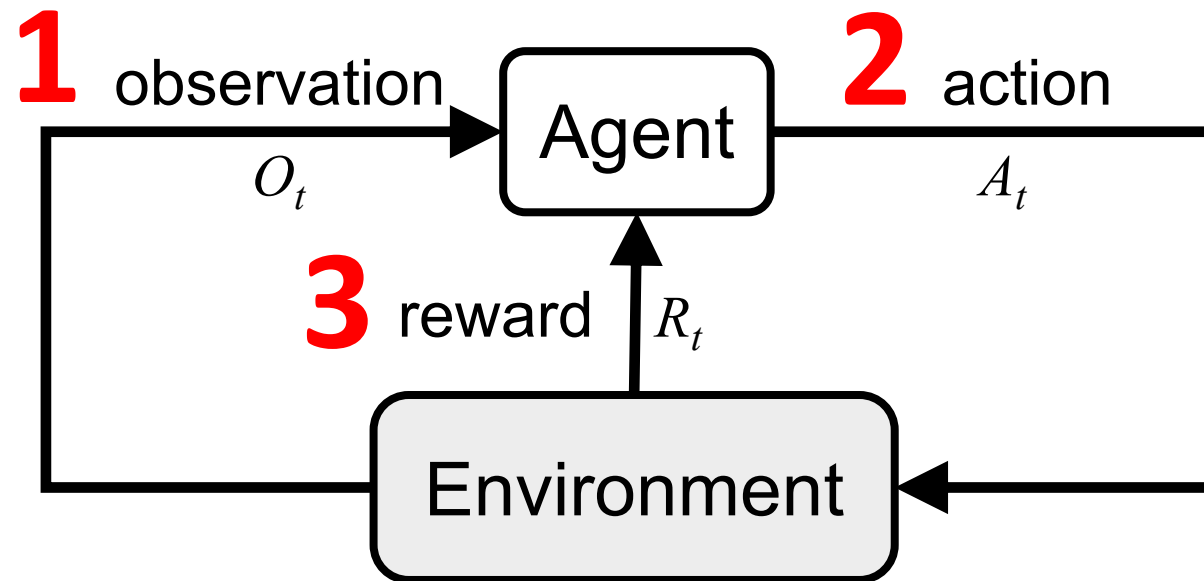
Agent

Environment

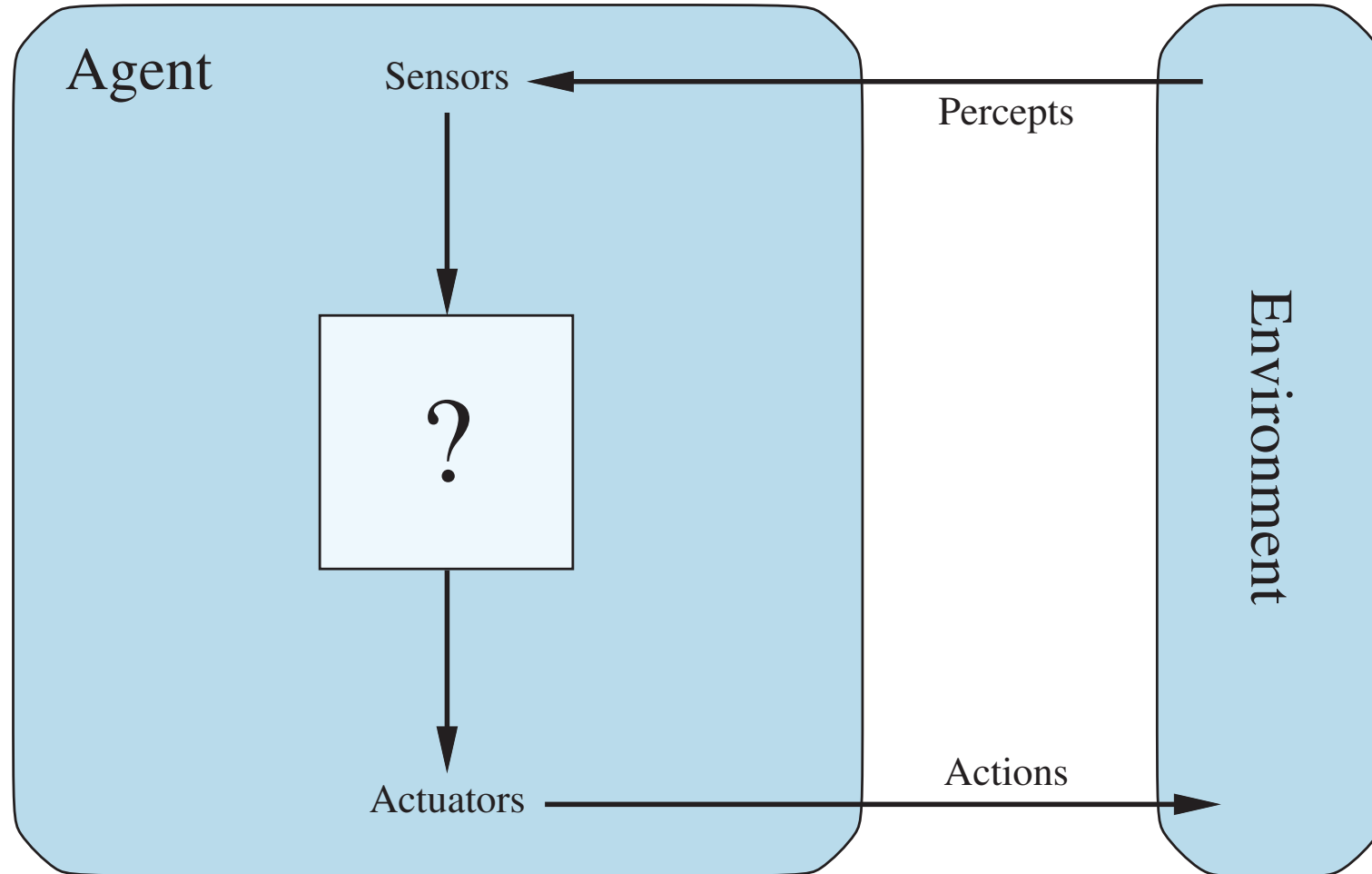
Reinforcement Learning (DL)



Reinforcement Learning (DL)



Agents interact with environments through sensors and actuators

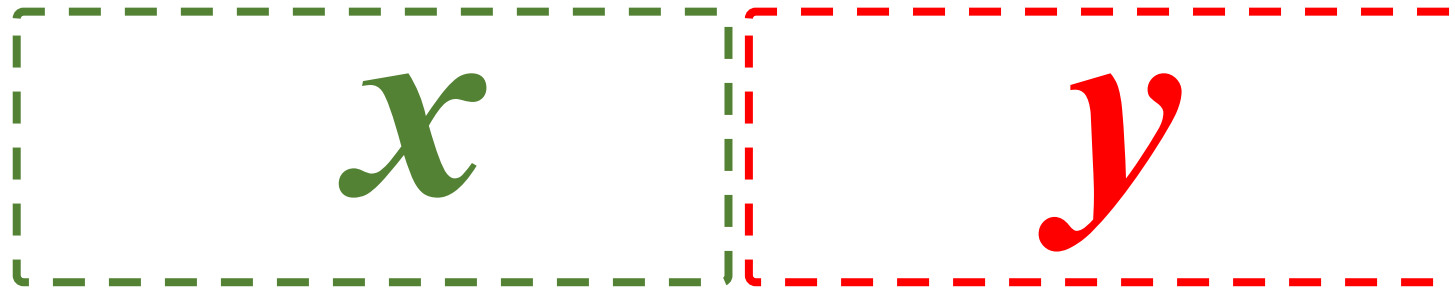


Machine Learning

Supervised Learning (Classification)

Learning from Examples

$$y = f(x)$$

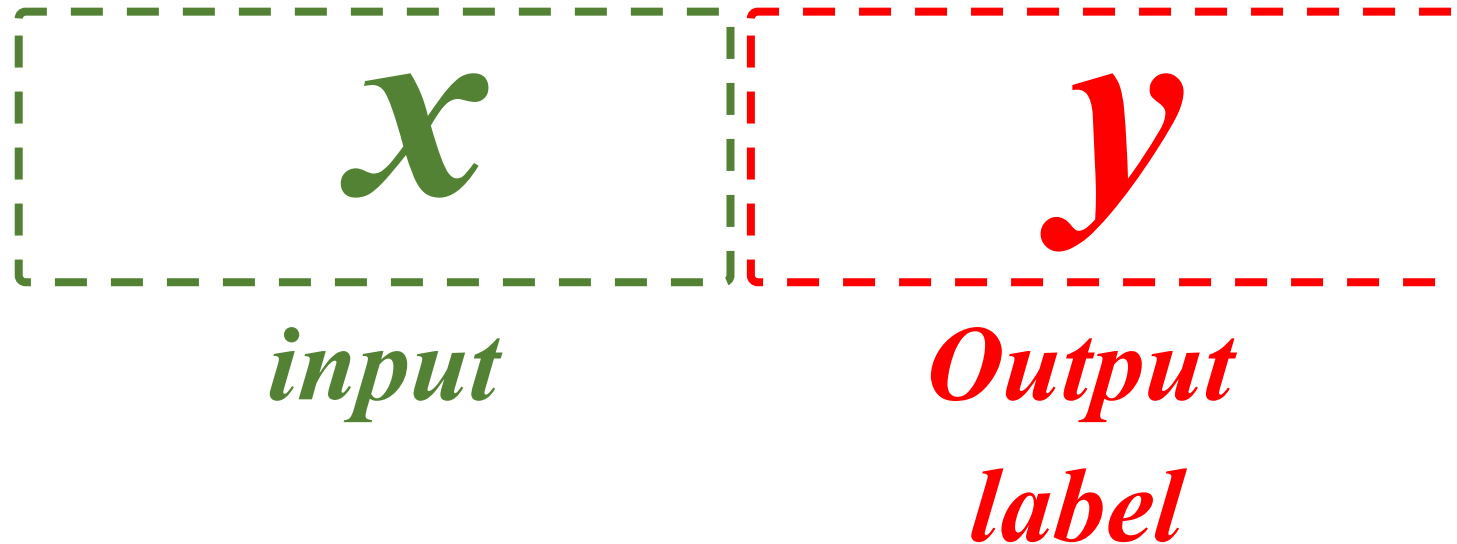


Machine Learning

Supervised Learning (Classification)

Learning from Examples

$$y = f(x)$$



Iris flower data set

setosa



versicolor



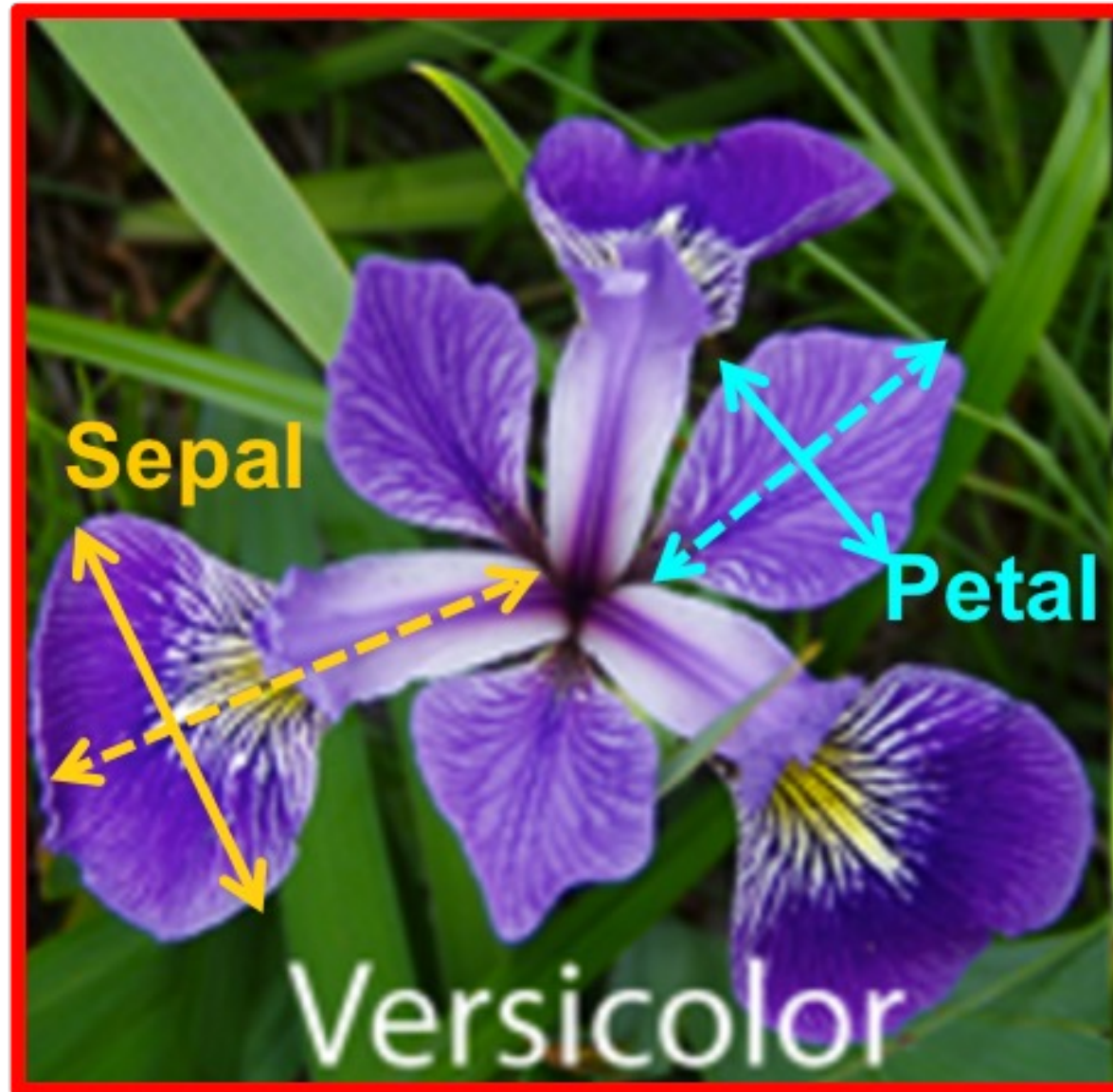
virginica



Source: https://en.wikipedia.org/wiki/Iris_flower_data_set

Source: <http://suruchifialoke.com/2016-10-13-machine-learning-tutorial-iris-classification/>

Iris Classification



iris.data

<https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data>

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.4,3.7,1.5,0.2,Iris-setosa
4.8,3.4,1.6,0.2,Iris-setosa
4.8,3.0,1.4,0.1,Iris-setosa
4.3,3.0,1.1,0.1,Iris-setosa
5.8,4.0,1.2,0.2,Iris-setosa
5.7,4.4,1.5,0.4,Iris-setosa
5.4,3.9,1.3,0.4,Iris-setosa
5.1,3.5,1.4,0.3,Iris-setosa
5.7,3.8,1.7,0.3,Iris-setosa
5.1,3.8,1.5,0.3,Iris-setosa
5.4,3.4,1.7,0.2,Iris-setosa
5.1,3.7,1.5,0.4,Iris-setosa
4.6,3.6,1.0,0.2,Iris-setosa
5.1,3.3,1.7,0.5,Iris-setosa
4.8,3.4,1.9,0.2,Iris-setosa
5.0,3.0,1.6,0.2,Iris-setosa
5.0,3.4,1.6,0.4,Iris-setosa
```

setosa



virginica



versicolor



Machine Learning

Supervised Learning (Classification)

Learning from Examples

$$y = f(x)$$

```
5.1, 3.5, 1.4, 0.2, Iris-setosa
4.9, 3.0, 1.4, 0.2, Iris-setosa
4.7, 3.2, 1.3, 0.2, Iris-setosa
7.0, 3.2, 4.7, 1.4, Iris-versicolor
6.4, 3.2, 4.5, 1.5, Iris-versicolor
6.9, 3.1, 4.9, 1.5, Iris-versicolor
6.3, 3.3, 6.0, 2.5, Iris-virginica
5.8, 2.7, 5.1, 1.9, Iris-virginica
7.1, 3.0, 5.9, 2.1, Iris-virginica
```

Machine Learning

Supervised Learning (Classification)

Learning from Examples

$$y = f(x)$$

Example

5.1	3.5	1.4	0.2	Iris-setosa
4.9	3.0	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
7.0	3.2	4.7	1.4	Iris-versicolor
6.4	3.2	4.5	1.5	Iris-versicolor
6.9	3.1	4.9	1.5	Iris-versicolor
6.3	3.3	6.0	2.5	Iris-virginica
5.8	2.7	5.1	1.9	Iris-virginica
7.1	3.0	5.9	2.1	Iris-virginica

Machine Learning

Supervised Learning (Classification)

Learning from Examples

$$y = f(x)$$

Example

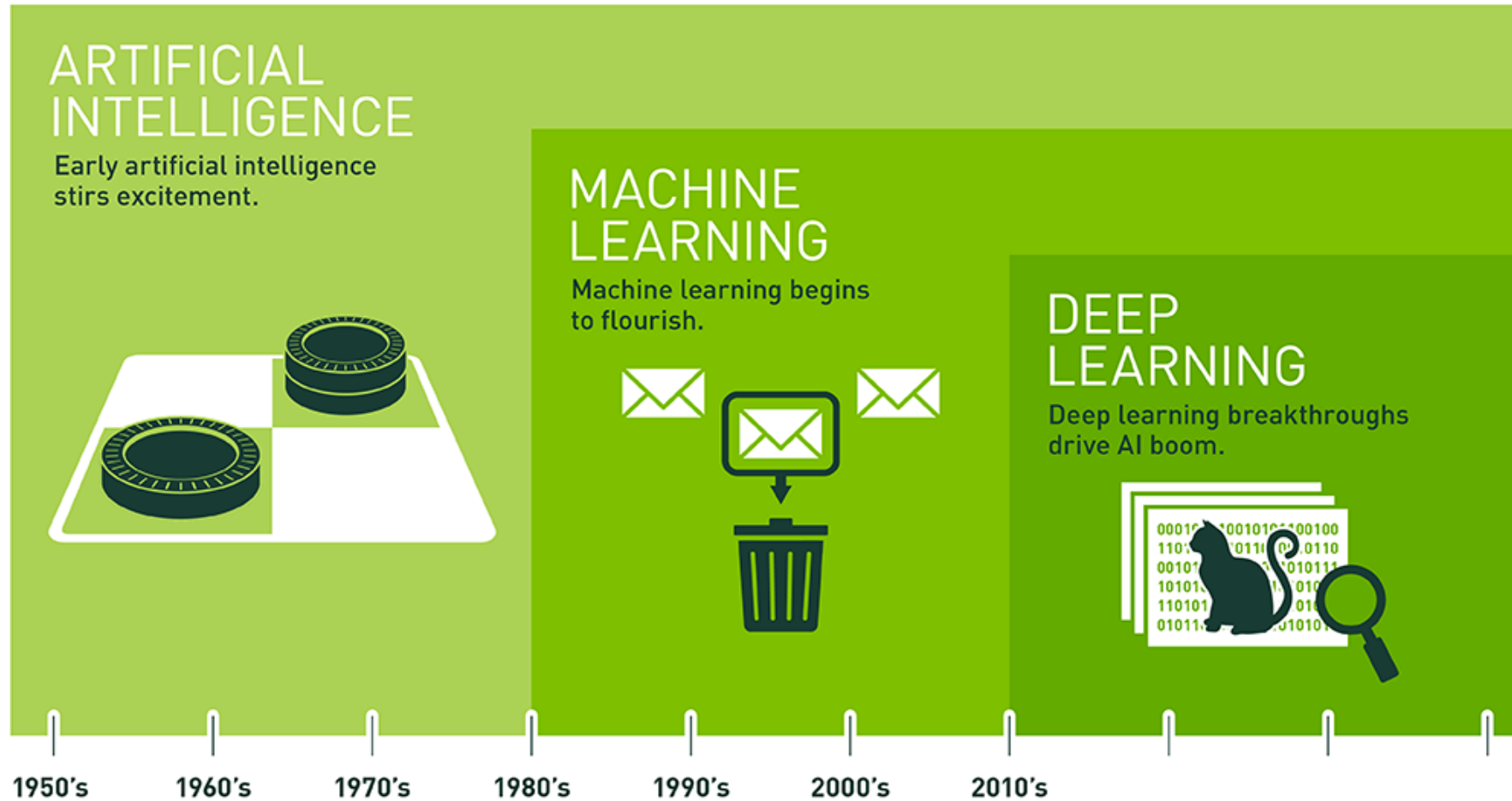
x

5.1, 3.5, 1.4, 0.2	Iris-setosa
4.9, 3.0, 1.4, 0.2	Iris-setosa
4.7, 3.2, 1.3, 0.2	Iris-setosa
7.0, 3.2, 4.7, 1.4	Iris-versicolor
6.4, 3.2, 4.5, 1.5	Iris-versicolor
6.9, 3.1, 4.9, 1.5	Iris-versicolor
6.3, 3.3, 6.0, 2.5	Iris-virginica
5.8, 2.7, 5.1, 1.9	Iris-virginica
7.1, 3.0, 5.9, 2.1	Iris-virginica

y

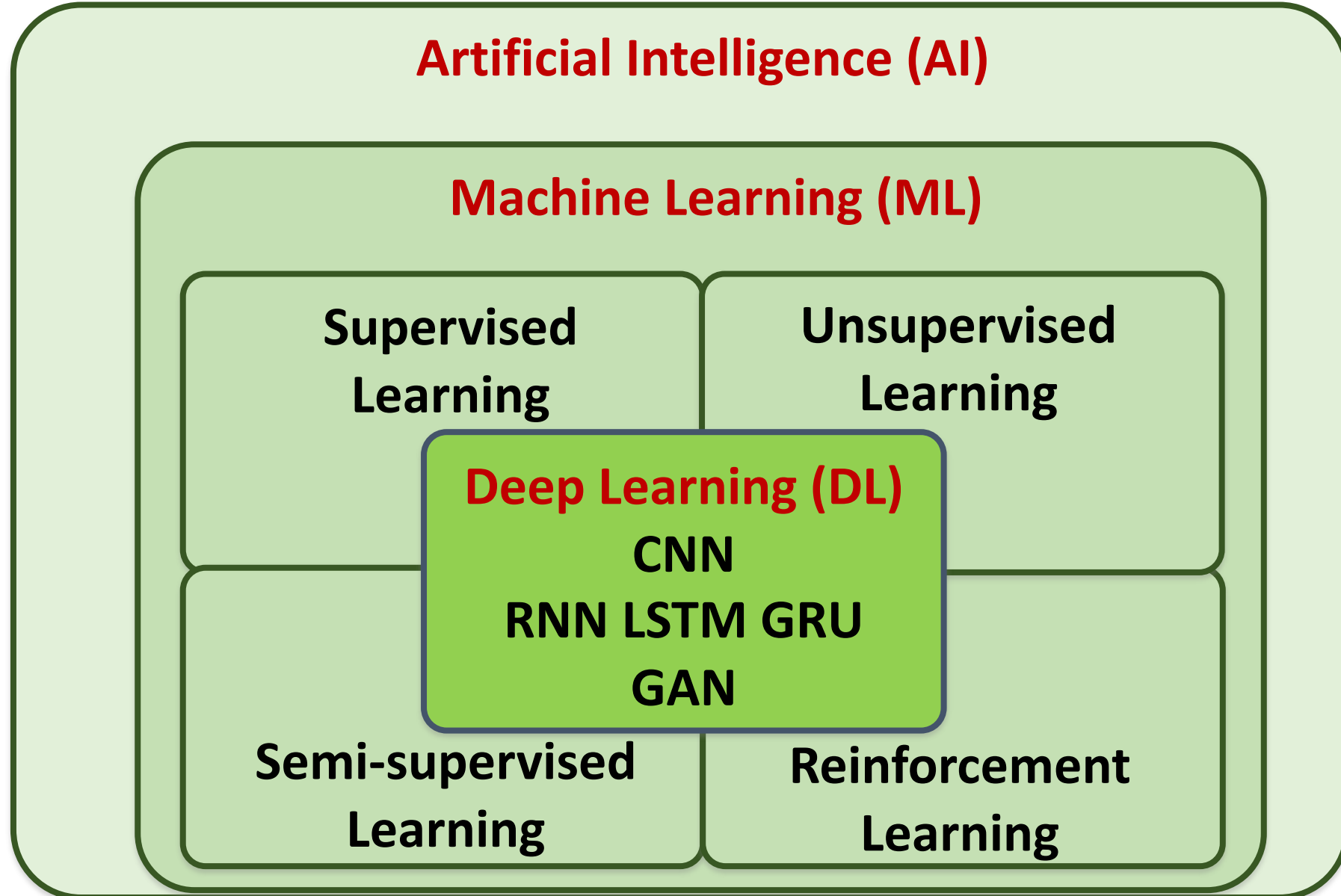
Artificial Intelligence

Machine Learning & Deep Learning

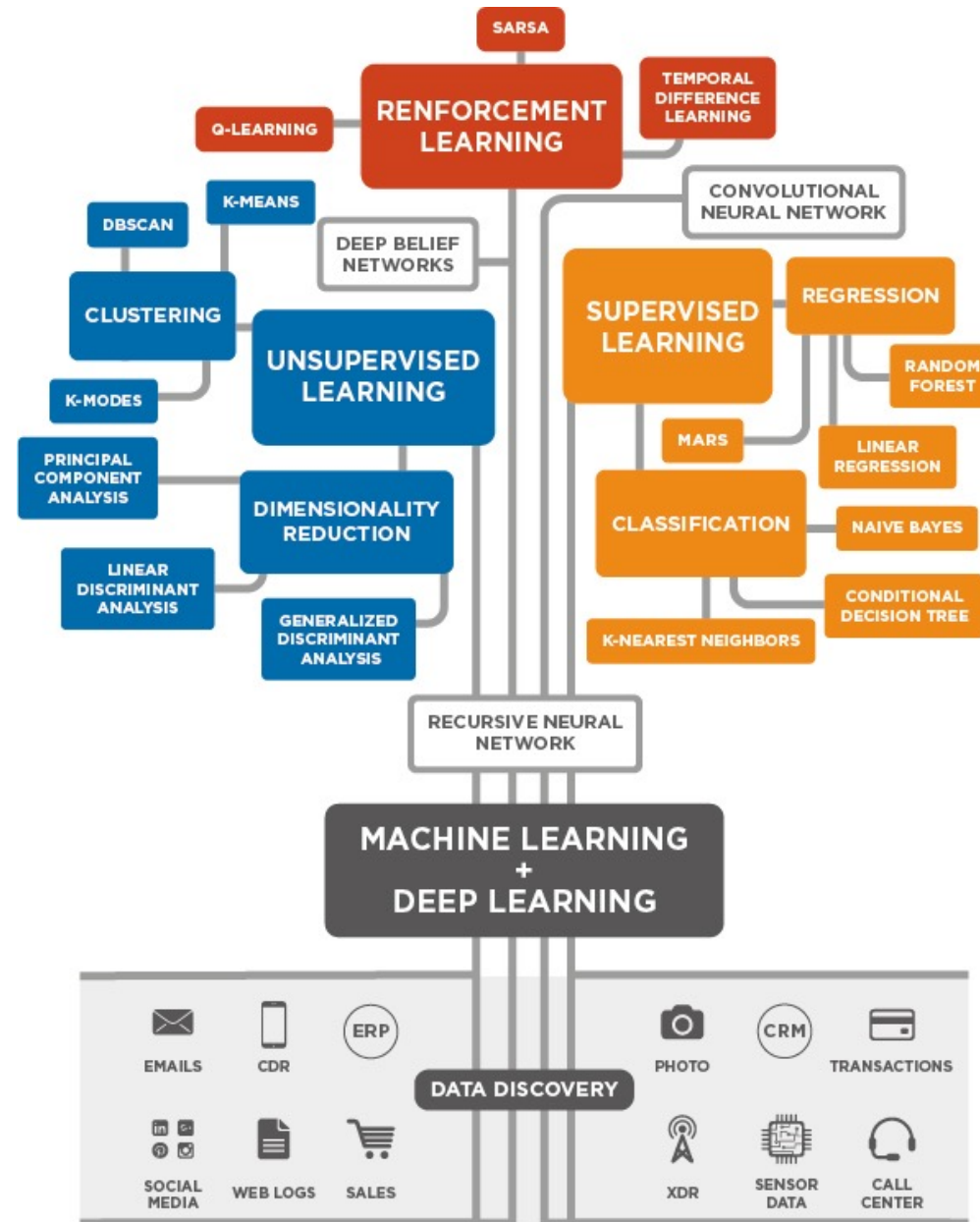


Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

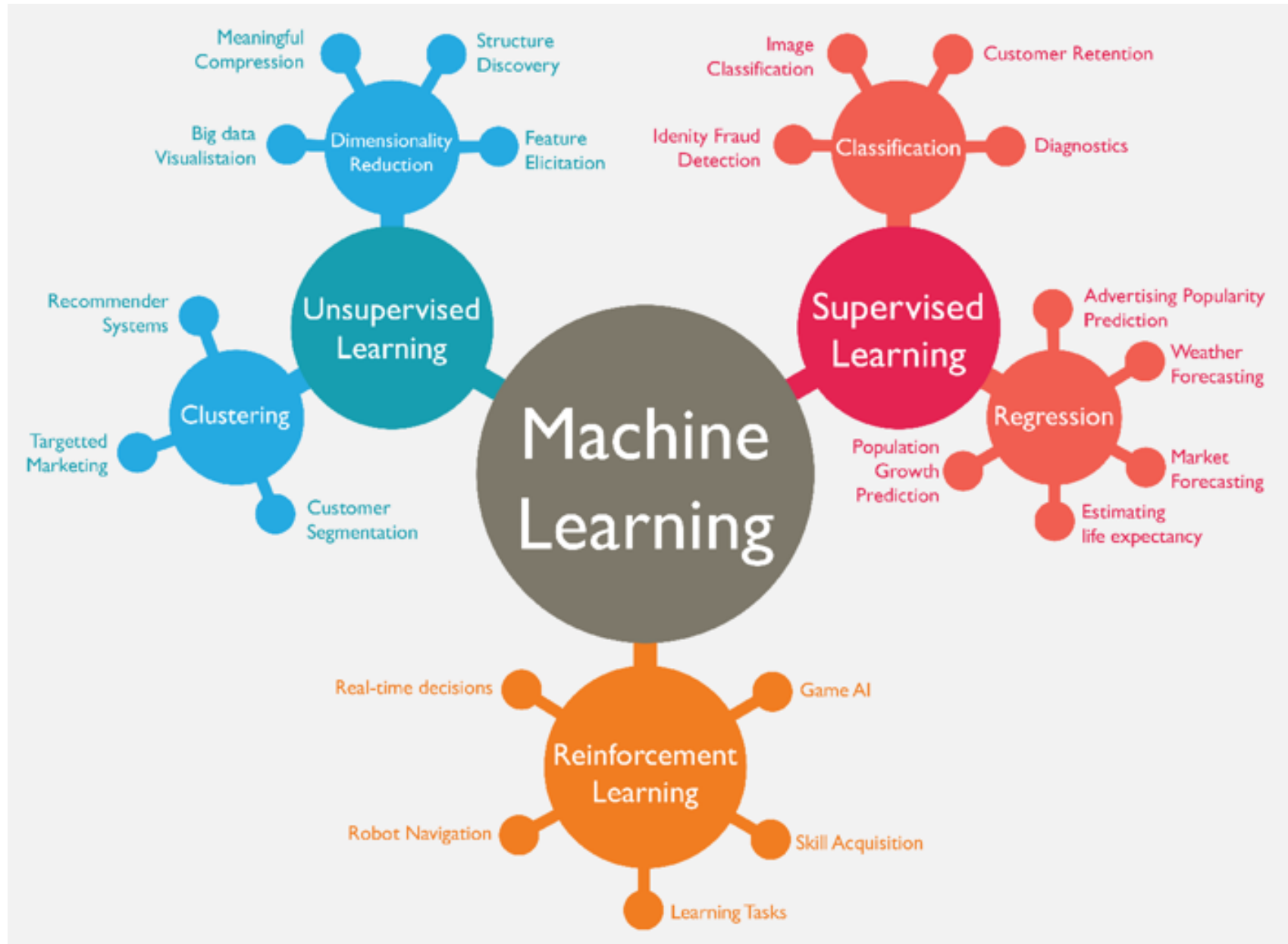
AI, ML, DL



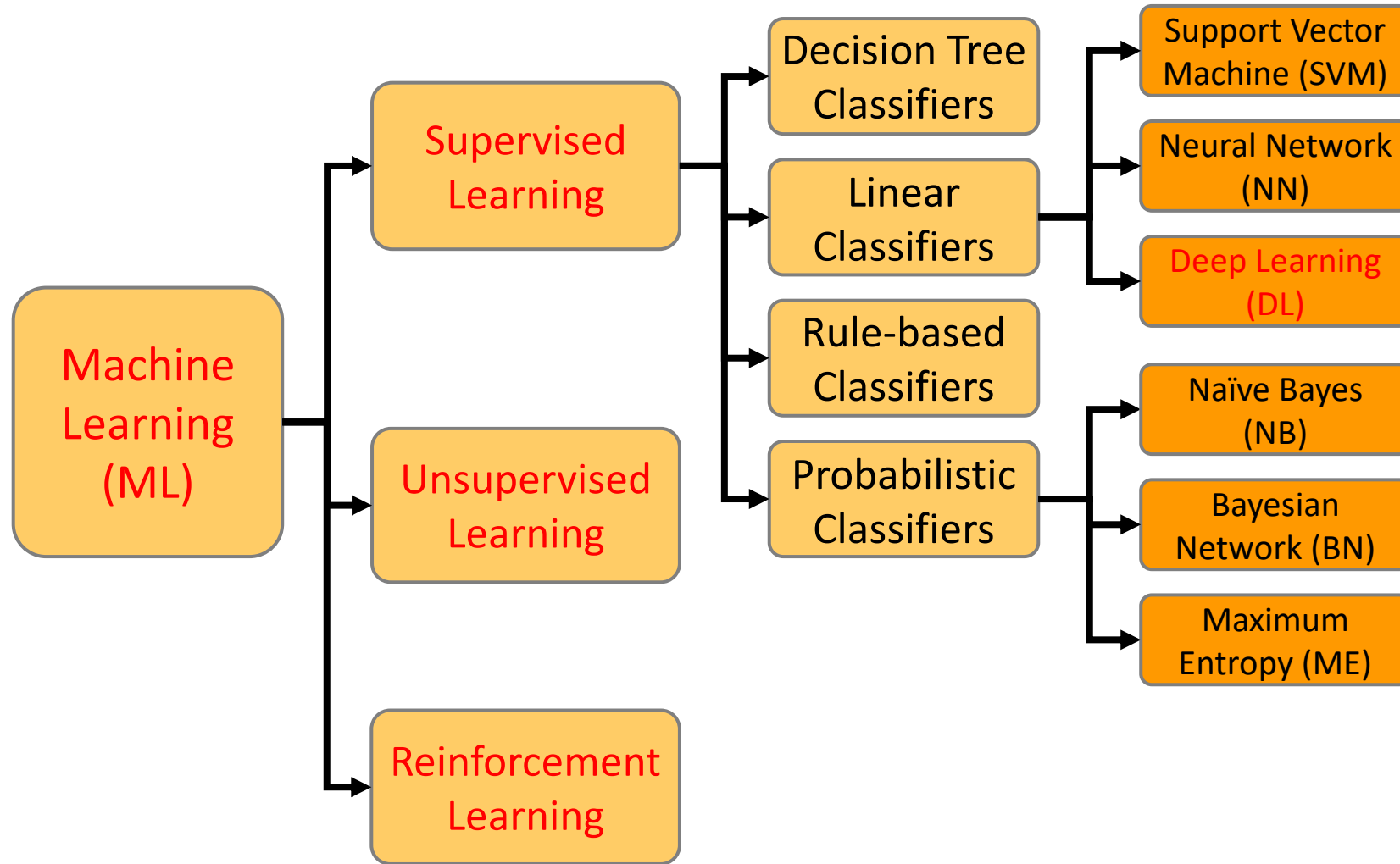
3 Machine Learning Algorithms



Machine Learning (ML)



Machine Learning (ML) / Deep Learning (DL)



Machine Learning Models

Deep Learning

Association rules

Decision tree

Clustering

Bayesian

Kernel

Ensemble

Dimensionality reduction

Regression Analysis

Instance based

Machine Learning: Data Mining Tasks & Methods

Data Mining Tasks & Methods	Data Mining Algorithms	Learning Type
Prediction		
Classification	Decision Trees, Neural Networks, Support Vector Machines, kNN, Naïve Bayes, GA	Supervised
Regression	Linear/Nonlinear Regression, ANN, Regression Trees, SVM, kNN, GA	Supervised
Time series	Autoregressive Methods, Averaging Methods, Exponential Smoothing, ARIMA	Supervised
Association		
Market-basket	Apriori, OneR, ZeroR, Eclat, GA	Unsupervised
Link analysis	Expectation Maximization, Apriori Algorithm, Graph-Based Matching	Unsupervised
Sequence analysis	Apriori Algorithm, FP-Growth, Graph-Based Matching	Unsupervised
Segmentation		
Clustering	k-means, Expectation Maximization (EM)	Unsupervised
Outlier analysis	k-means, Expectation Maximization (EM)	Unsupervised

Data Mining Methods

- **Supervised Learning**
 - **Classification**
 - Class Label Prediction
 - **Regression**
 - Numeric Value Prediction
- **Unsupervised Learning**
 - **Clustering**
 - **Association**

Scikit-Learn

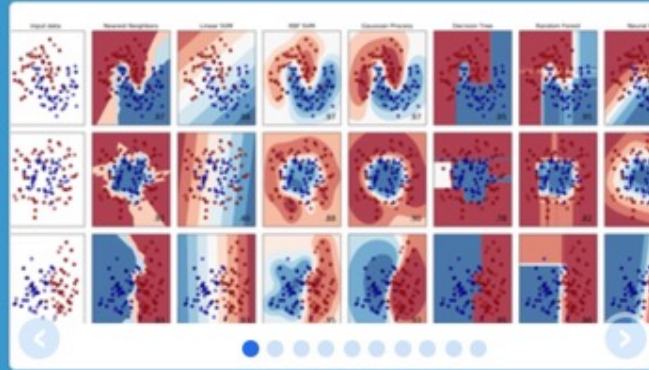
Machine Learning in Python

Scikit-Learn



Home Installation Documentation ▾ Examples

Google Custom Search



scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.

Algorithms: SVM, nearest neighbors, random forest, ... — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

Algorithms: SVR, ridge regression, Lasso, ... — Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes

Algorithms: k-Means, spectral clustering, mean-shift, ... — Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency

Algorithms: PCA, feature selection, non-negative matrix factorization. — Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning

Modules: grid search, cross validation, metrics. — Examples

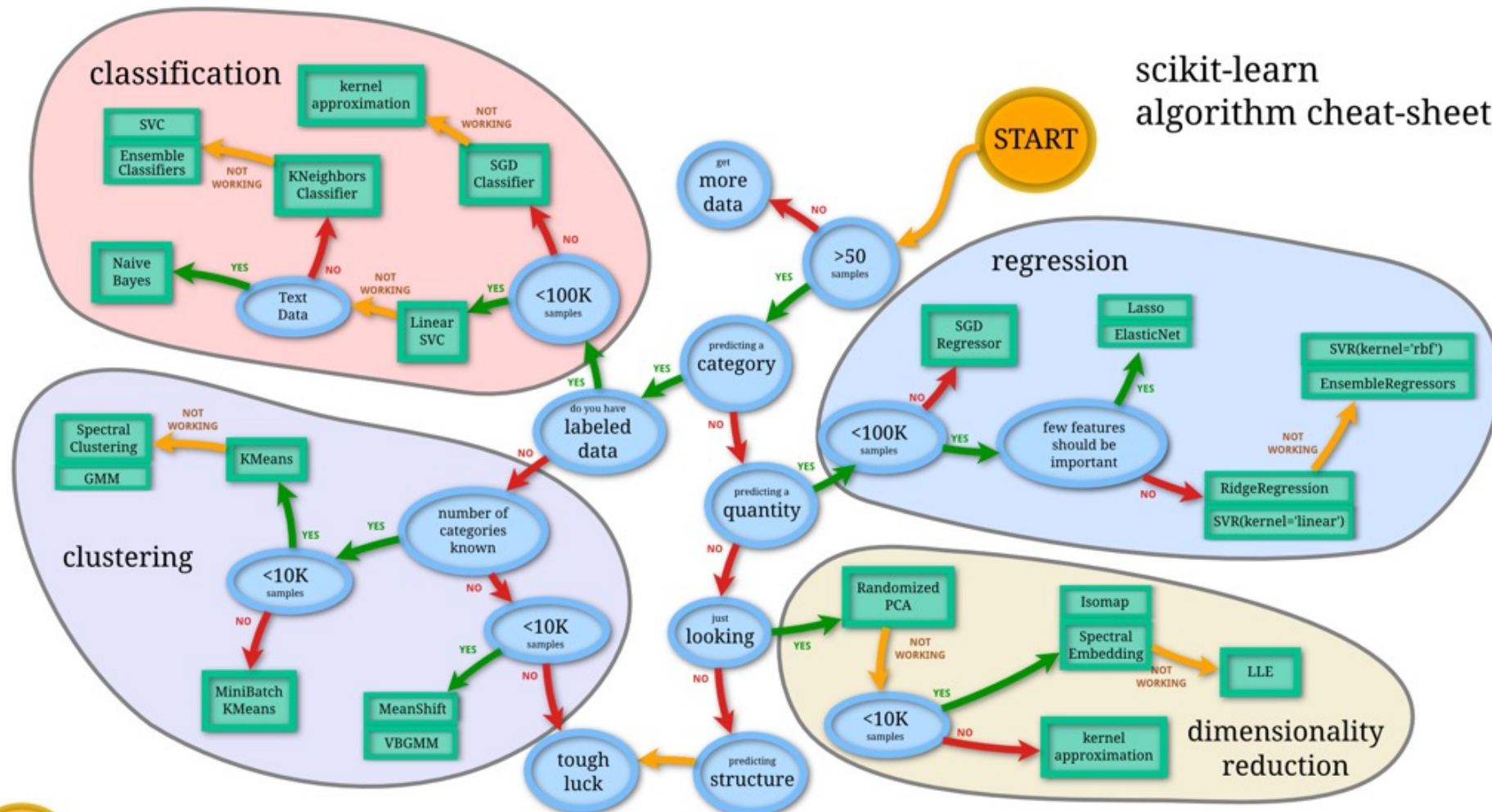
Preprocessing

Feature extraction and normalization.

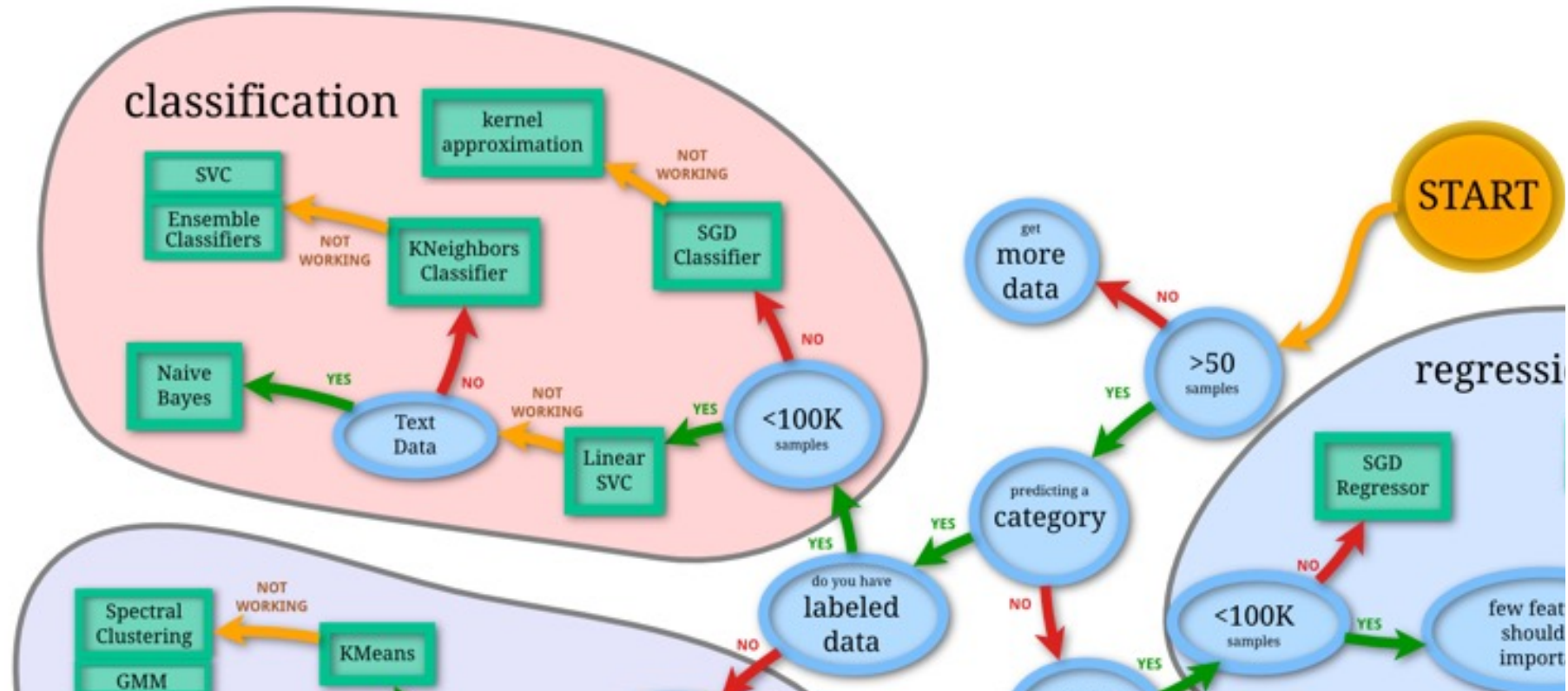
Application: Transforming input data such as text for use with machine learning algorithms.

Modules: preprocessing, feature extraction. — Examples

Scikit-Learn Machine Learning Map

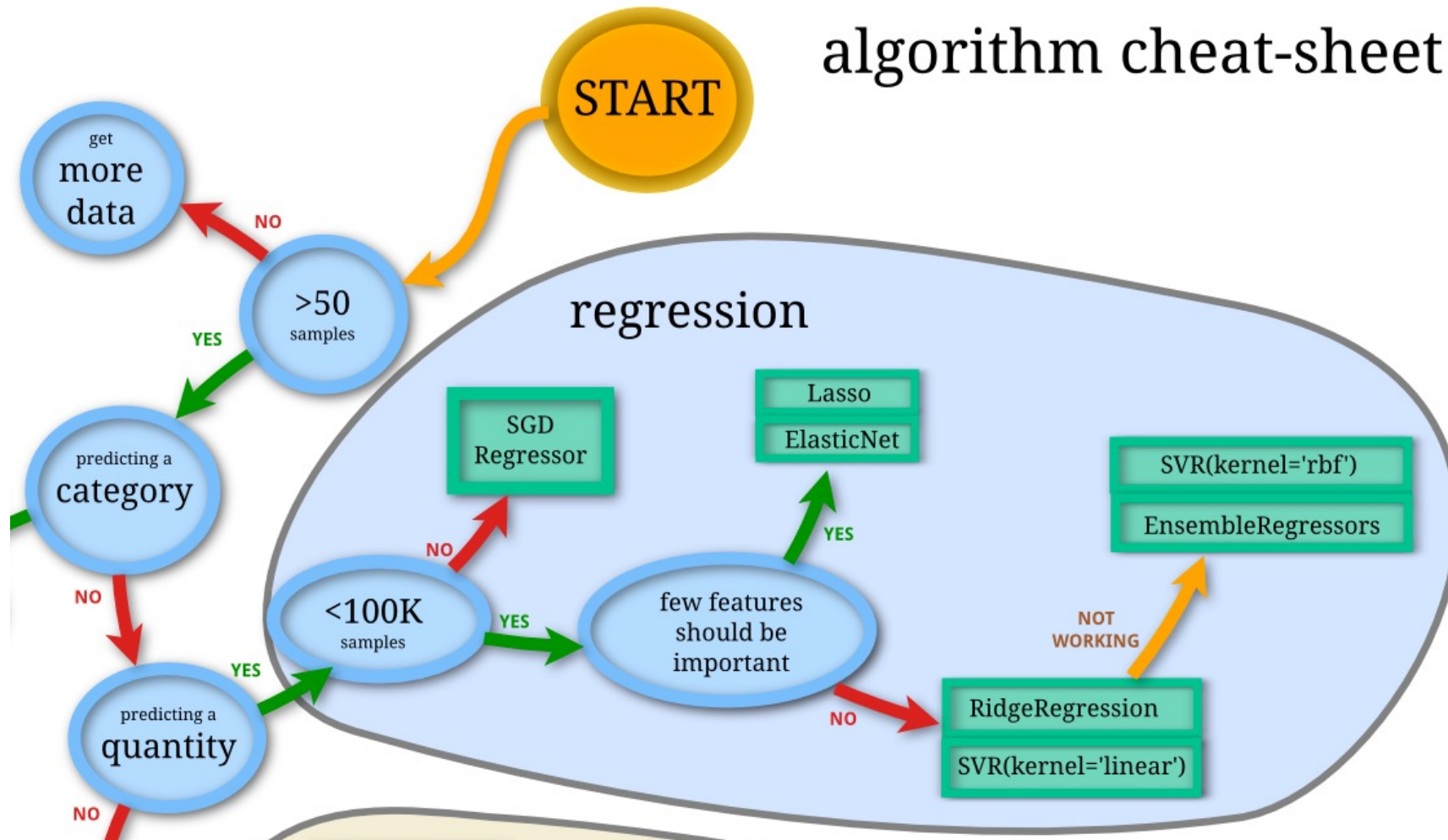


Scikit-Learn Machine Learning Map

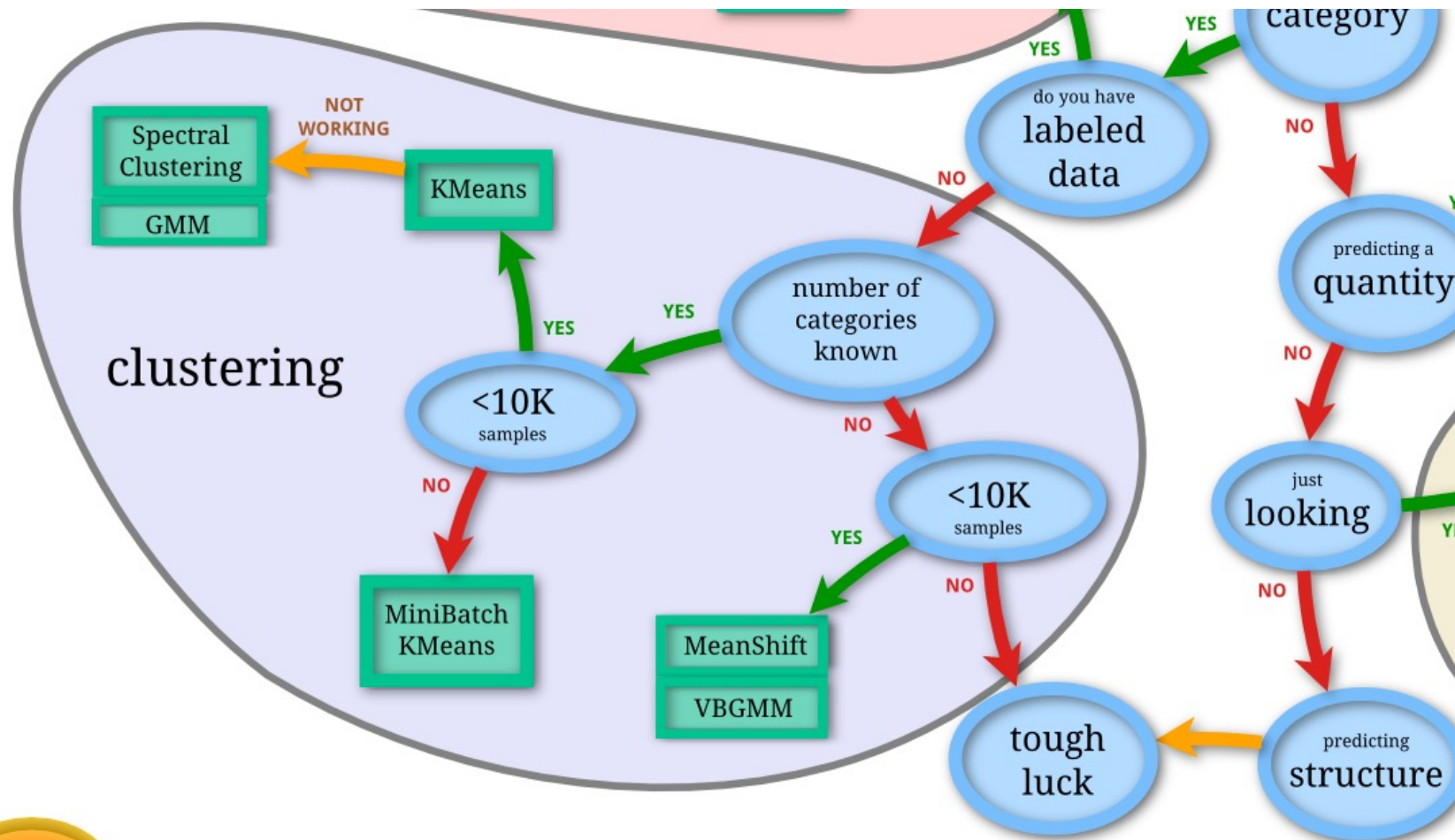


Scikit-Learn Machine Learning Map

scikit-learn
algorithm cheat-sheet



Scikit-Learn Machine Learning Map



Iris flower data set

setosa



versicolor



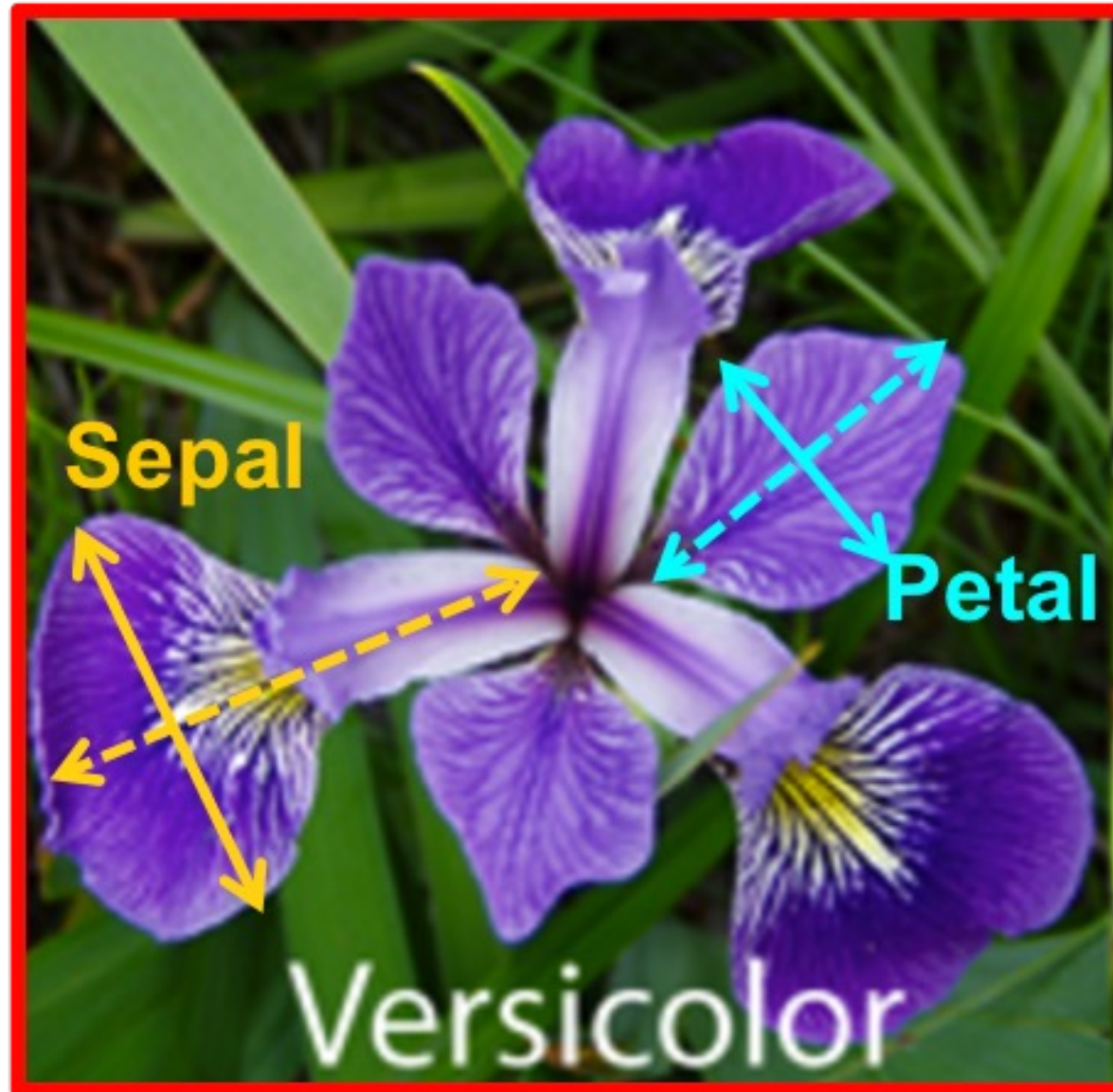
virginica



Source: https://en.wikipedia.org/wiki/Iris_flower_data_set

Source: <http://suruchifialoke.com/2016-10-13-machine-learning-tutorial-iris-classification/>

Iris Classification



iris.data

<https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data>

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.4,3.7,1.5,0.2,Iris-setosa
4.8,3.4,1.6,0.2,Iris-setosa
4.8,3.0,1.4,0.1,Iris-setosa
4.3,3.0,1.1,0.1,Iris-setosa
5.8,4.0,1.2,0.2,Iris-setosa
5.7,4.4,1.5,0.4,Iris-setosa
5.4,3.9,1.3,0.4,Iris-setosa
5.1,3.5,1.4,0.3,Iris-setosa
5.7,3.8,1.7,0.3,Iris-setosa
5.1,3.8,1.5,0.3,Iris-setosa
5.4,3.4,1.7,0.2,Iris-setosa
5.1,3.7,1.5,0.4,Iris-setosa
4.6,3.6,1.0,0.2,Iris-setosa
5.1,3.3,1.7,0.5,Iris-setosa
4.8,3.4,1.9,0.2,Iris-setosa
5.0,3.0,1.6,0.2,Iris-setosa
5.0,3.4,1.6,0.4,Iris-setosa
```

setosa



virginica



versicolor



Machine Learning

Supervised Learning (Classification)

Learning from Examples

```
5.1, 3.5, 1.4, 0.2, Iris-setosa  
4.9, 3.0, 1.4, 0.2, Iris-setosa  
4.7, 3.2, 1.3, 0.2, Iris-setosa  
7.0, 3.2, 4.7, 1.4, Iris-versicolor  
6.4, 3.2, 4.5, 1.5, Iris-versicolor  
6.9, 3.1, 4.9, 1.5, Iris-versicolor  
6.3, 3.3, 6.0, 2.5, Iris-virginica  
5.8, 2.7, 5.1, 1.9, Iris-virginica  
7.1, 3.0, 5.9, 2.1, Iris-virginica
```


Machine Learning

Supervised Learning (Classification)

Learning from Examples

$$y = f(x)$$

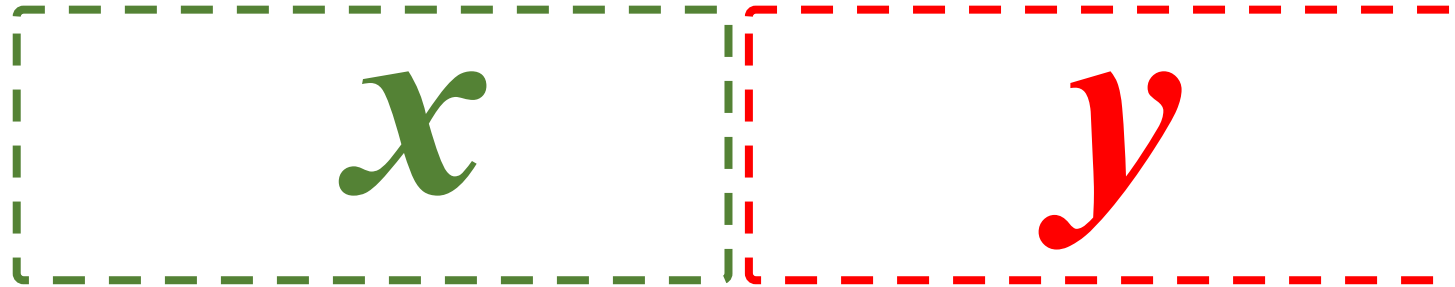
x	5.1, 3.5, 1.4, 0.2	Iris-setosa	y
	4.9, 3.0, 1.4, 0.2	Iris-setosa	
	4.7, 3.2, 1.3, 0.2	Iris-setosa	
	7.0, 3.2, 4.7, 1.4	Iris-versicolor	
	6.4, 3.2, 4.5, 1.5	Iris-versicolor	
	6.9, 3.1, 4.9, 1.5	Iris-versicolor	
	6.3, 3.3, 6.0, 2.5	Iris-virginica	
	5.8, 2.7, 5.1, 1.9	Iris-virginica	
	7.1, 3.0, 5.9, 2.1	Iris-virginica	

Machine Learning

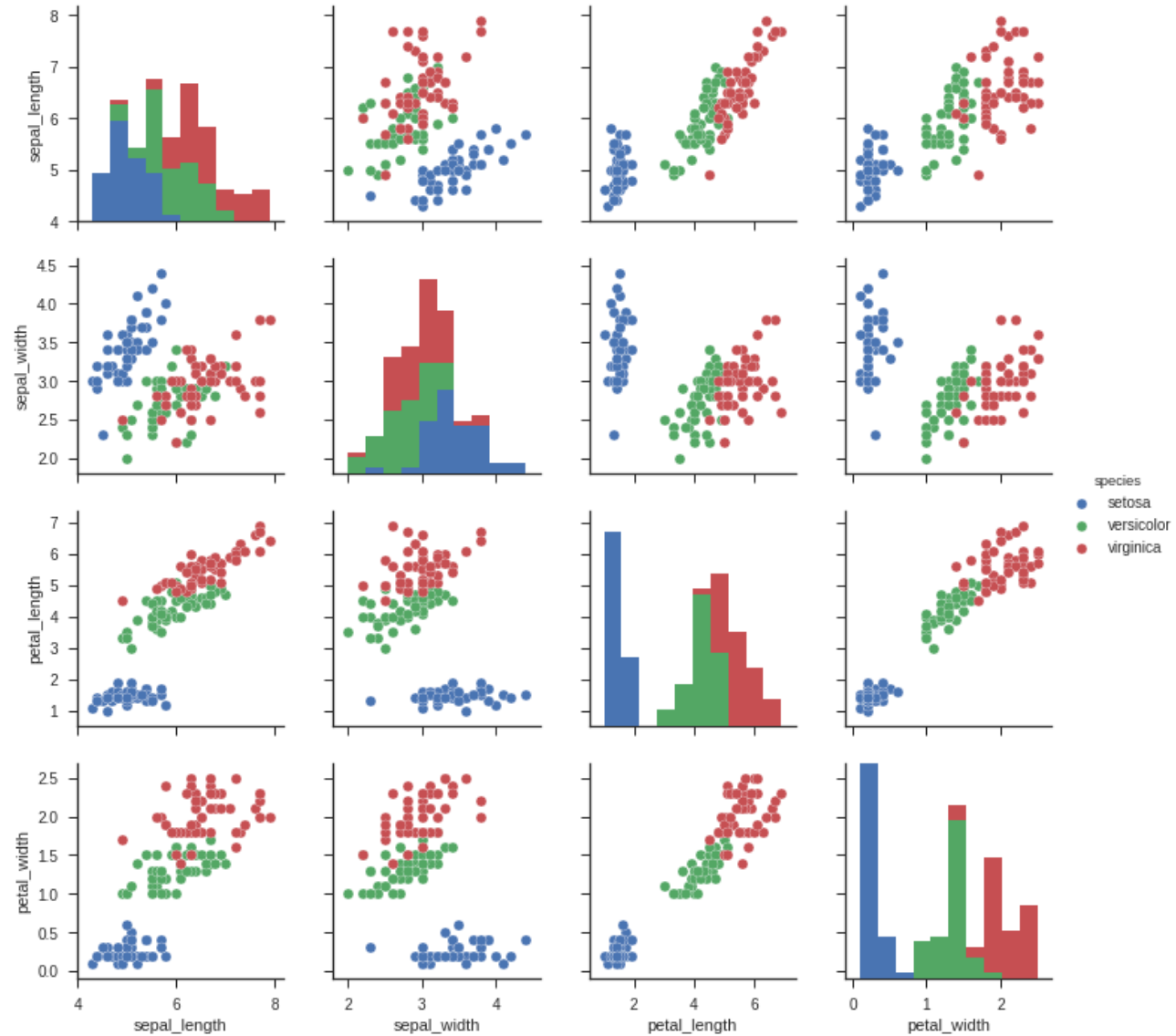
Supervised Learning (Classification)

Learning from Examples

$$y = f(x)$$



Iris Data Visualization



Python in Google Colab (Python101)

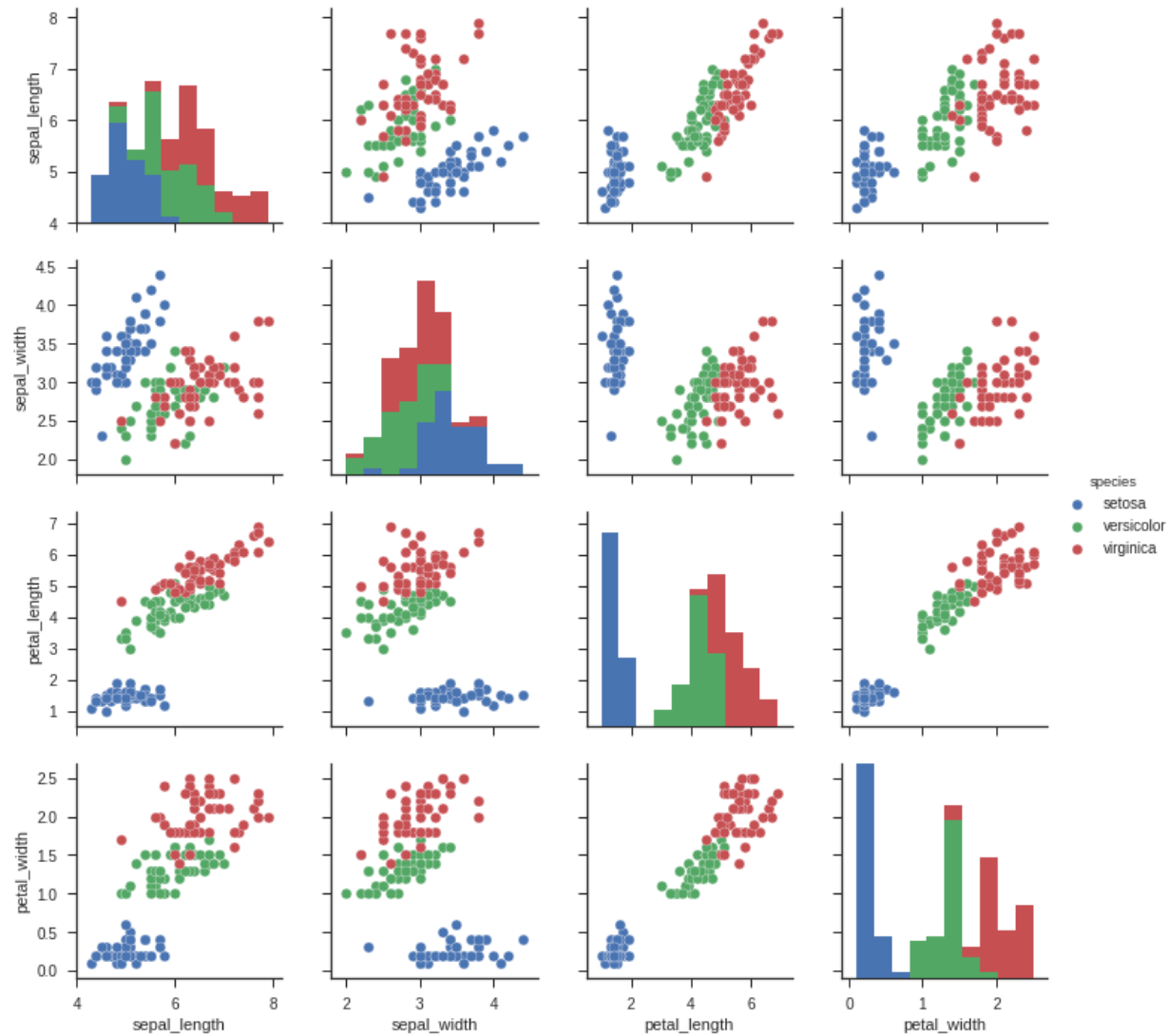
<https://colab.research.google.com/drive/1FEG6DnGvwfUbeo4zJ1zTunjMqf2RkCrT>

The screenshot displays the Google Colab interface for a notebook titled "python101.ipynb". The top navigation bar includes "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help", with a "Last saved at 10:43 AM" timestamp. On the right, there are icons for "Comment", "Share", "Settings", and a user profile. A "Table of contents" sidebar on the left lists various topics, with "Classification and Prediction" selected. The main workspace shows a code cell with the following Python code:

```
1 # Import libraries
2 import numpy as np
3 import pandas as pd
4 %matplotlib inline
5 import matplotlib.pyplot as plt
6 import seaborn as sns
7 from pandas.plotting import scatter_matrix
8
9 # Import sklearn
10 from sklearn import model_selection
11 from sklearn.metrics import classification_report
12 from sklearn.metrics import confusion_matrix
13 from sklearn.metrics import accuracy_score
14 from sklearn.linear_model import LogisticRegression
15 from sklearn.tree import DecisionTreeClassifier
16 from sklearn.neighbors import KNeighborsClassifier
17 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
18 from sklearn.naive_bayes import GaussianNB
19 from sklearn.svm import SVC
20 from sklearn.neural_network import MLPClassifier
21 print("Imported")
22
23 # Load dataset
24 url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
25 names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
```

<https://tinyurl.com/aintpupython101>

```
import seaborn as sns
sns.set(style="ticks", color_codes=True)
iris = sns.load_dataset("iris")
g = sns.pairplot(iris, hue="species")
```



```
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from pandas.plotting import scatter_matrix
```

```
# Import Libraries
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from pandas.plotting import scatter_matrix
print('imported')
```

imported

```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"  
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']  
df = pd.read_csv(url, names=names)  
print(df.head(10))
```

```
# Load dataset  
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"  
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']  
df = pd.read_csv(url, names=names)  
print(df.head(10)).
```

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa

df.tail(10)

```
print(df.tail(10)).
```

	sepal-length	sepal-width	petal-length	petal-width	class
140	6.7	3.1	5.6	2.4	Iris-virginica
141	6.9	3.1	5.1	2.3	Iris-virginica
142	5.8	2.7	5.1	1.9	Iris-virginica
143	6.8	3.2	5.9	2.3	Iris-virginica
144	6.7	3.3	5.7	2.5	Iris-virginica
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

df.describe()

```
print(df.describe())
```

	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```
print(df.info())  
print(df.shape)
```

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 150 entries, 0 to 149  
Data columns (total 5 columns):  
sepal-length      150 non-null float64  
sepal-width       150 non-null float64  
petal-length      150 non-null float64  
petal-width       150 non-null float64  
class             150 non-null object  
dtypes: float64(4), object(1)  
memory usage: 5.9+ KB  
None
```

```
print(df.shape)
```

```
(150, 5)
```

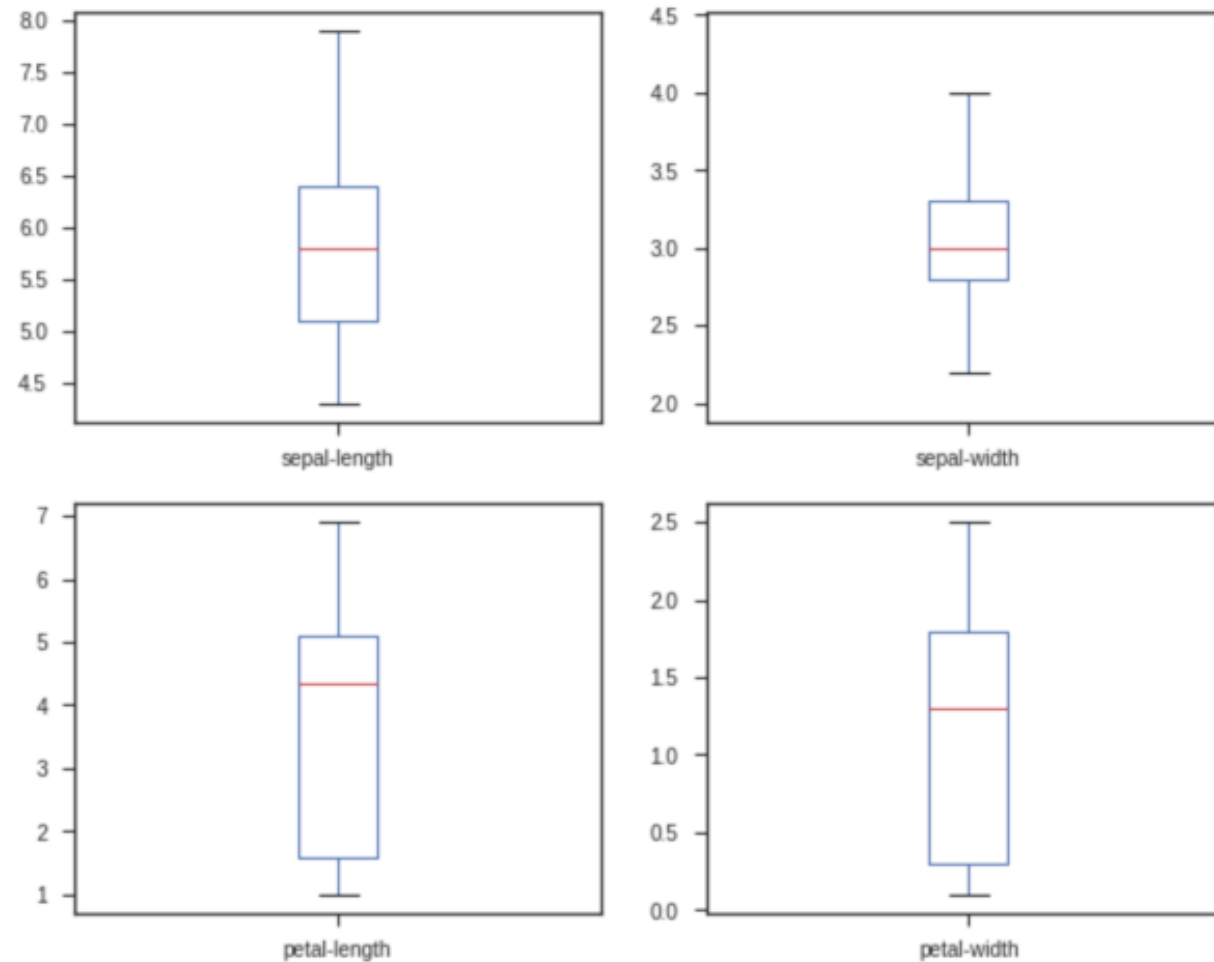
```
df.groupby('class').size()
```

```
print(df.groupby('class').size())
```

```
class
Iris-setosa      50
Iris-versicolor  50
Iris-virginica   50
dtype: int64
```

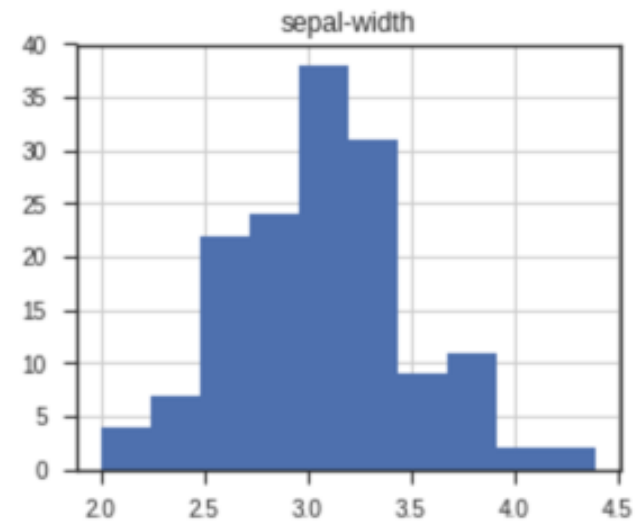
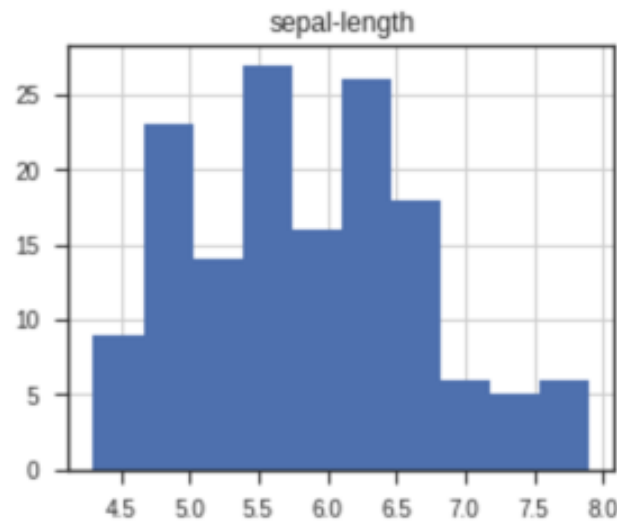
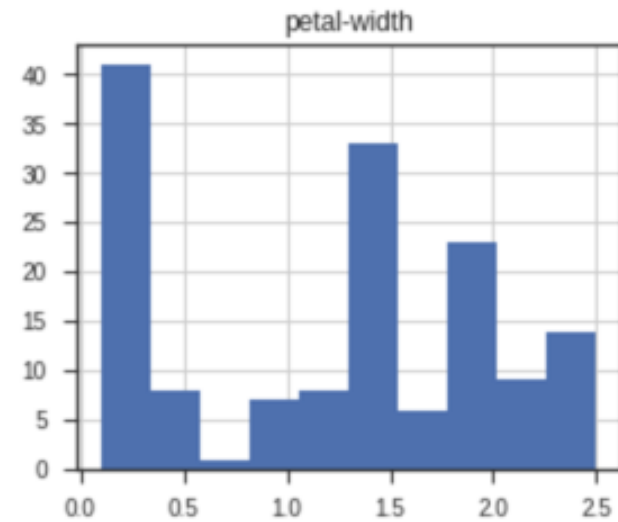
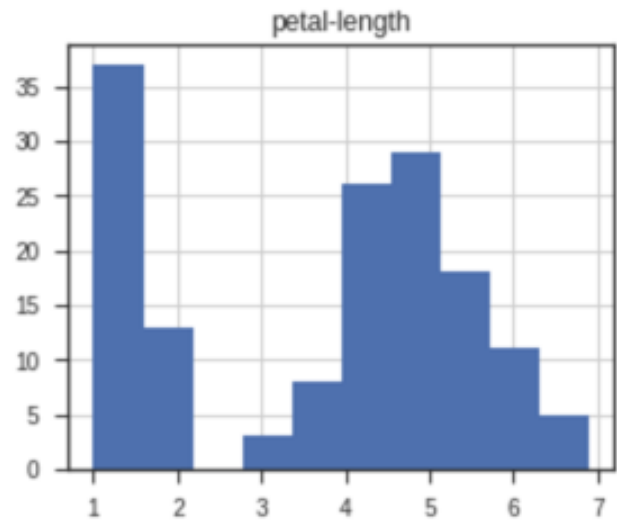
```
plt.rcParams["figure.figsize"] = (10,8)
df.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
plt.show()
```

```
plt.rcParams["figure.figsize"] = (10,8)
df.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
plt.show().
```



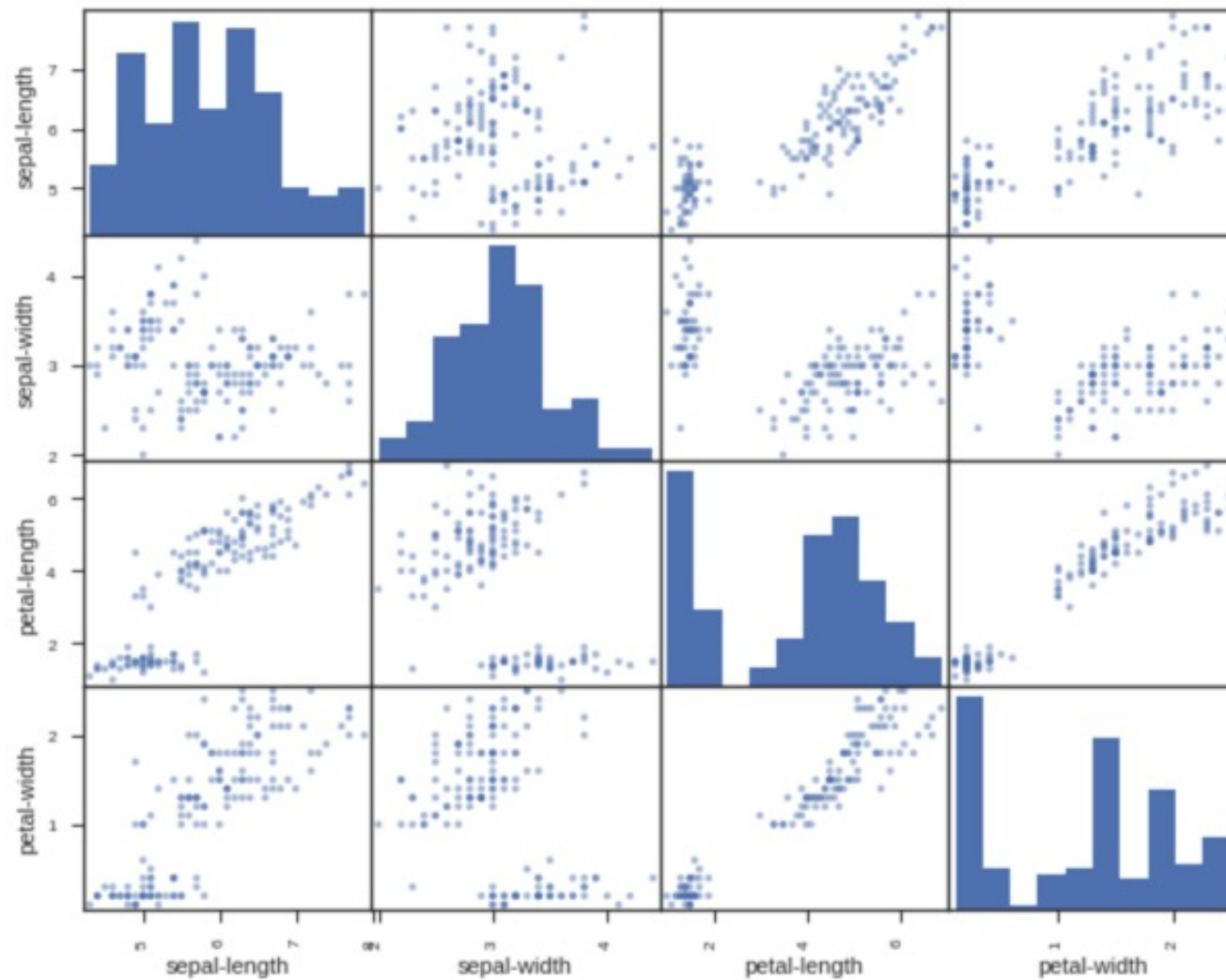
```
df.hist()  
plt.show()
```

```
df.hist()  
plt.show()
```



```
scatter_matrix(df)
plt.show()
```

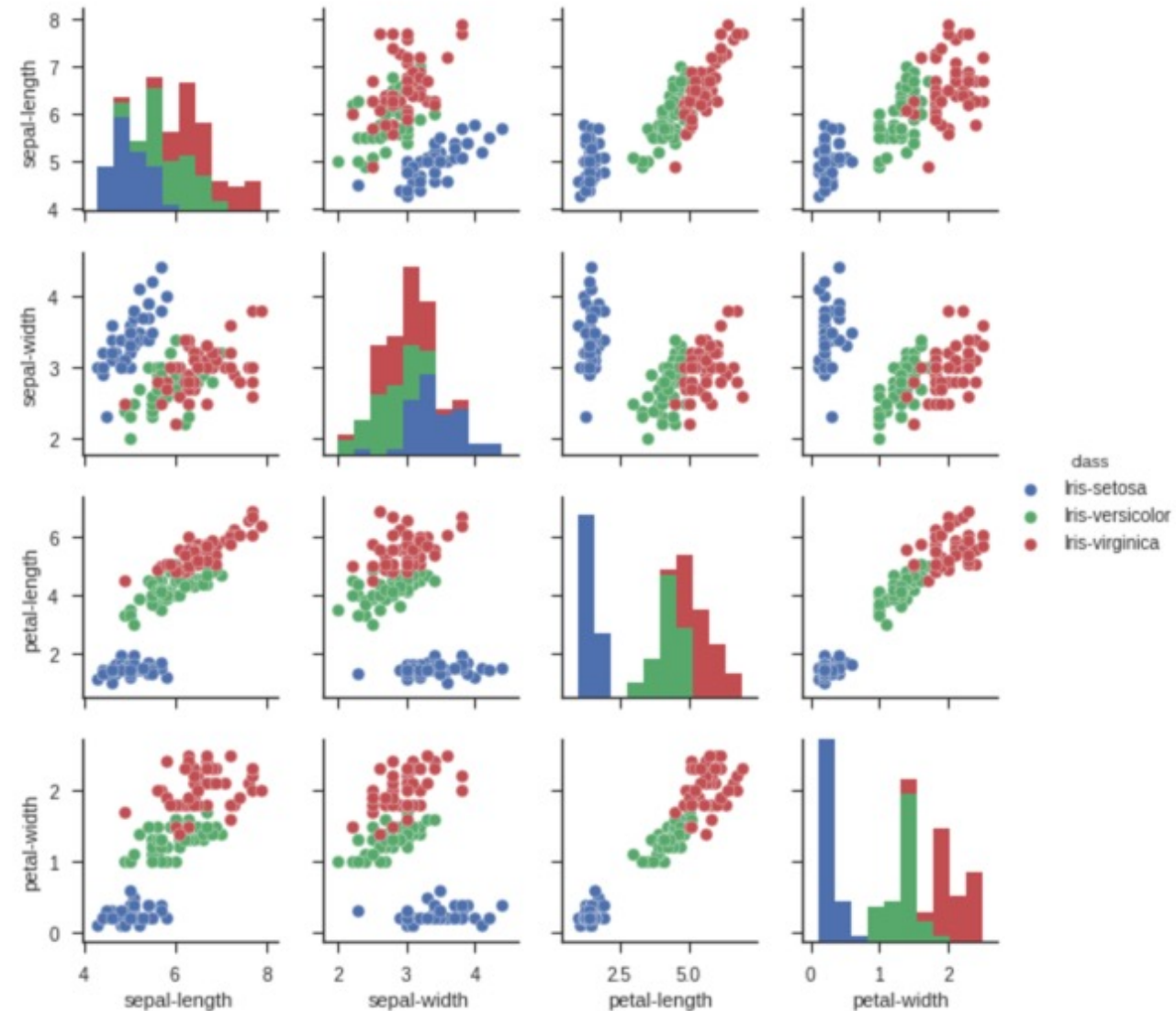
```
scatter_matrix(df)
plt.show()
```



```
sns.pairplot(df, hue="class", size=2)
```

```
sns.pairplot(df, hue="class", size=2)
```

```
<seaborn.axisgrid.PairGrid at 0x7f1d21267390>
```



**Machine Learning:
Supervised Learning:
Classification
and
Prediction**

Machine Learning: Data Mining Tasks & Methods

**Prediction
Classification**

**Supervised Learning:
Classification
and
Prediction**

Data Mining Tasks & Methods	Data Mining Algorithms	Learning Type
Prediction		
Classification	Decision Trees, Neural Networks, Support Vector Machines, kNN, Naïve Bayes, GA	Supervised
Regression	Linear/Nonlinear Regression, ANN, Regression Trees, SVM, kNN, GA	Supervised
Time series	Autoregressive Methods, Averaging Methods, Exponential Smoothing, ARIMA	Supervised
Association		
Market-basket	Apriori, OneR, ZeroR, Eclat, GA	Unsupervised
Link analysis	Expectation Maximization, Apriori Algorithm, Graph-Based Matching	Unsupervised
Sequence analysis	Apriori Algorithm, FP-Growth, Graph-Based Matching	Unsupervised
Segmentation		
Clustering	k-means, Expectation Maximization (EM)	Unsupervised
Outlier analysis	k-means, Expectation Maximization (EM)	Unsupervised

Machine Learning: Supervised Learning Classification and Prediction

The image shows a Jupyter Notebook interface for a file named 'python101.ipynb'. The top navigation bar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help', with a timestamp 'Last saved at 10:43 AM'. On the right, there are icons for 'Comment', 'Share', and a user profile 'A'. A sidebar on the left contains a 'Table of contents' with a search icon and a list of topics: 'Machine Learning with scikit-learn', 'Classification and Prediction', 'K-Means Clustering', 'Deep Learning for Financial Time Series Forecasting', 'Portfolio Optimization and Algorithmic Trading', 'Investment Portfolio Optimisation with Python', 'Efficient Frontier Portfolio Optimisation in Python', 'Investment Portfolio Optimization', 'Text Analytics and Natural Language Processing (NLP)', 'Python for Natural Language Processing', 'spaCy Chinese Model', 'Open Chinese Convert (OpenCC, 開放中文轉換)', 'Jieba 結巴中文分詞', 'Natural Language Toolkit (NLTK)', 'Stanza: A Python NLP Library for Many Human Languages', and 'Text Processing and Understanding'. The main area shows a code cell with the following Python code:

```
1 # Import libraries
2 import numpy as np
3 import pandas as pd
4 %matplotlib inline
5 import matplotlib.pyplot as plt
6 import seaborn as sns
7 from pandas.plotting import scatter_matrix
8
9 # Import sklearn
10 from sklearn import model_selection
11 from sklearn.metrics import classification_report
12 from sklearn.metrics import confusion_matrix
13 from sklearn.metrics import accuracy_score
14 from sklearn.linear_model import LogisticRegression
15 from sklearn.tree import DecisionTreeClassifier
16 from sklearn.neighbors import KNeighborsClassifier
17 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
18 from sklearn.naive_bayes import GaussianNB
19 from sklearn.svm import SVC
20 from sklearn.neural_network import MLPClassifier
21 print("Imported")
22
23 # Load dataset
24 url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
25 names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
```

```
# Import sklearn
from sklearn import model_selection
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
print("Imported")
```



```
1 # Load dataset
2 url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
3 names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
4 df = pd.read_csv(url, names=names)
5
6 print(df.head(10))
7 print(df.tail(10))
8 print(df.describe())
9 print(df.info())
10 print(df.shape)
11 print(df.groupby('class').size())
12
13 plt.rcParams["figure.figsize"] = (10,8)
14 df.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
15 plt.show()
16
17 df.hist()
18 plt.show()
19
20 scatter_matrix(df)
21 plt.show()
22
23 sns.pairplot(df, hue="class", size=2).
```

```
☐→
```

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa
	sepal-length	sepal-width	petal-length	petal-width	class
140	6.7	3.1	5.6	2.4	Iris-virginica
141	6.9	3.1	5.1	2.3	Iris-virginica
142	5.8	2.7	5.1	1.9	Iris-virginica



```
1 # Load dataset
2 url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
3 names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
4 df = pd.read_csv(url, names=names)
5
6 print(df.head(10))
7 print(df.tail(10))
8 print(df.describe())
9 print(df.info())
10 print(df.shape)
11 print(df.groupby('class').size())
12
13 plt.rcParams["figure.figsize"] = (10,8)
14 df.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
15 plt.show()
16
17 df.hist()
18 plt.show()
19
20 scatter_matrix(df)
21 plt.show()
22
23 sns.pairplot(df, hue="class", size=2).
```

```
[ ]>      sepal-length  sepal-width  petal-length  petal-width  class
0         5.1         3.5         1.4         0.2  Iris-setosa
1         4.9         3.0         1.4         0.2  Iris-setosa
2         4.7         3.2         1.3         0.2  Iris-setosa
3         4.6         3.1         1.5         0.2  Iris-setosa
4         5.0         3.6         1.4         0.2  Iris-setosa
5         5.4         3.9         1.7         0.4  Iris-setosa
6         4.6         3.4         1.4         0.3  Iris-setosa
7         5.0         3.4         1.5         0.2  Iris-setosa
8         4.4         2.9         1.4         0.2  Iris-setosa
9         4.9         3.1         1.5         0.1  Iris-setosa
      sepal-length  sepal-width  petal-length  petal-width  class
140         6.7         3.1         5.6         2.4  Iris-virginica
141         6.9         3.1         5.1         2.3  Iris-virginica
142         5.8         2.7         5.1         1.9  Iris-virginica
```

df.corr()

```
1 df.corr(.
```

	sepal-length	sepal-width	petal-length	petal-width
sepal-length	1.000000	-0.109369	0.871754	0.817954
sepal-width	-0.109369	1.000000	-0.420516	-0.356544
petal-length	0.871754	-0.420516	1.000000	0.962757
petal-width	0.817954	-0.356544	0.962757	1.000000

```
# Split-out validation dataset
array = df.values
X = array[:,0:4]
Y = array[:,4]
validation_size = 0.20
seed = 7
X_train, X_validation, Y_train, Y_validation =
model_selection.train_test_split(X, Y,
test_size=validation_size, random_state=seed)
scoring = 'accuracy'
```

```
1 # Split-out validation dataset
2 array = df.values
3 X = array[:,0:4]
4 Y = array[:,4]
5 validation_size = 0.20
6 seed = 7
7 X_train, X_validation, Y_train, Y_validation = model_selection.train_test_split(X, Y, test_size=validation_size, random_state=seed)
8 scoring = 'accuracy'
```

```
1 len(Y_validation).
```

30

```
# Models  
models = []  
models.append(('LR', LogisticRegression()))  
models.append(('LDA',  
LinearDiscriminantAnalysis()))  
models.append(('KNN', KNeighborsClassifier()))  
models.append(('DT',  
DecisionTreeClassifier()))  
models.append(('NB', GaussianNB()))  
models.append(('SVM', SVC()))
```



```
# evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits=10,
random_state=seed)
    cv_results =
model_selection.cross_val_score(model,
X_train, Y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %.4f (%.4f)" % (name,
cv_results.mean(), cv_results.std())
    print(msg)
```

```

1 # Models
2 models = []
3 models.append(('LR', LogisticRegression()))
4 models.append(('LDA', LinearDiscriminantAnalysis()))
5 models.append(('KNN', KNeighborsClassifier()))
6 models.append(('DT', DecisionTreeClassifier()))
7 models.append(('NB', GaussianNB()))
8 models.append(('SVM', SVC()))
9 # evaluate each model in turn
10 results = []
11 names = []
12 for name, model in models:
13     kfold = model_selection.KFold(n_splits=10, random_state=seed)
14     cv_results = model_selection.cross_val_score(model, X_train, Y_train, cv=kfold, scoring=scoring)
15     results.append(cv_results)
16     names.append(name)
17     msg = "%s: %.4f (%.4f)" % (name, cv_results.mean(), cv_results.std())
18     print(msg)

```

```

LR: 0.9667 (0.0408)
LDA: 0.9750 (0.0382)
KNN: 0.9833 (0.0333)
DT: 0.9750 (0.0382)
NB: 0.9750 (0.0534)
SVM: 0.9917 (0.0250)

```

```
# Make predictions on validation dataset
model = KNeighborsClassifier()
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)
print("%.4f" % accuracy_score(Y_validation,
predictions))
print(confusion_matrix(Y_validation,
predictions))
print(classification_report(Y_validation,
predictions))
print(model)
```

```

1 # Make predictions on validation dataset
2 model = KNeighborsClassifier()
3 model.fit(X_train, Y_train)
4 predictions = model.predict(X_validation)
5 print("%.4f" % accuracy_score(Y_validation, predictions))
6 print(confusion_matrix(Y_validation, predictions))
7 print(classification_report(Y_validation, predictions))
8 print(model)

```

0.9000

```

[[ 7  0  0]
 [ 0 11  1]
 [ 0  2  9]]

```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	7
Iris-versicolor	0.85	0.92	0.88	12
Iris-virginica	0.90	0.82	0.86	11
avg / total	0.90	0.90	0.90	30

```

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=1, n_neighbors=5, p=2,
                    weights='uniform')

```

```
# Make predictions on validation dataset
model = SVC()
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)
print("%.4f" % accuracy_score(Y_validation,
predictions))
print(confusion_matrix(Y_validation,
predictions))
print(classification_report(Y_validation,
predictions))
print(model)
```

```
model = SVC()
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)
```

```
1 # Make predictions on validation dataset
2 model = SVC()
3 model.fit(X_train, Y_train)
4 predictions = model.predict(X_validation)
5 print("%.4f" % accuracy_score(Y_validation, predictions))
6 print(confusion_matrix(Y_validation, predictions))
7 print(classification_report(Y_validation, predictions))
8 print(model)
```

0.9333

```
[[ 7  0  0]
 [ 0 10  2]
 [ 0  0 11]]
```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	7
Iris-versicolor	1.00	0.83	0.91	12
Iris-virginica	0.85	1.00	0.92	11
avg / total	0.94	0.93	0.93	30

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

```

1 # Make predictions on validation dataset
2 model = DecisionTreeClassifier()
3 model.fit(X_train, Y_train)
4 predictions = model.predict(X_validation)
5 print("%.4f" % accuracy_score(Y_validation, predictions))
6 print(confusion_matrix(Y_validation, predictions))
7 print(classification_report(Y_validation, predictions))
8 print(model)

```

0.9000

```

[[ 7  0  0]
 [ 0 11  1]
 [ 0  2  9]]

```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	7
Iris-versicolor	0.85	0.92	0.88	12
Iris-virginica	0.90	0.82	0.86	11
avg / total	0.90	0.90	0.90	30

```

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

```

```
1 # Make predictions on validation dataset
2 model = GaussianNB(.)
3 model.fit(X_train, Y_train)
4 predictions = model.predict(X_validation)
5 print("%.4f" % accuracy_score(Y_validation, predictions))
6 print(confusion_matrix(Y_validation, predictions))
7 print(classification_report(Y_validation, predictions))
8 print(model)
```

0.8333

```
[[7 0 0]
 [0 9 3]
 [0 2 9]]
```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	7
Iris-versicolor	0.82	0.75	0.78	12
Iris-virginica	0.75	0.82	0.78	11
avg / total	0.84	0.83	0.83	30

GaussianNB(priors=None)


```

1 # Make predictions on validation dataset
2 model = LogisticRegression()
3 model.fit(X_train, Y_train)
4 predictions = model.predict(X_validation)
5 print("%.4f" % accuracy_score(Y_validation, predictions))
6 print(confusion_matrix(Y_validation, predictions))
7 print(classification_report(Y_validation, predictions))
8 print(model)

```

0.8000

```

[[ 7  0  0]
 [ 0  7  5]
 [ 0  1 10]]

```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	7
Iris-versicolor	0.88	0.58	0.70	12
Iris-virginica	0.67	0.91	0.77	11
avg / total	0.83	0.80	0.80	30

```

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
verbose=0, warm_start=False)

```

```

1 # Make predictions on validation dataset
2 model = LinearDiscriminantAnalysis()
3 model.fit(X_train, Y_train)
4 predictions = model.predict(X_validation)
5 print("%.4f" % accuracy_score(Y_validation, predictions))
6 print(confusion_matrix(Y_validation, predictions))
7 print(classification_report(Y_validation, predictions))
8 print(model)

```

0.9667

```

[[ 7  0  0]
 [ 0 11  1]
 [ 0  0 11]]

```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	7
Iris-versicolor	1.00	0.92	0.96	12
Iris-virginica	0.92	1.00	0.96	11
avg / total	0.97	0.97	0.97	30

```

LinearDiscriminantAnalysis(n_components=None, priors=None, shrinkage=None,
                           solver='svd', store_covariance=False, tol=0.0001)

```

```

1 # Make predictions on validation dataset
2 model = MLPClassifier()
3 model.fit(X_train, Y_train)
4 predictions = model.predict(X_validation)
5 print("%.4f" % accuracy_score(Y_validation, predictions))
6 print(confusion_matrix(Y_validation, predictions))
7 print(classification_report(Y_validation, predictions))
8 print(model)

```

0.9000

```

[[ 7  0  0]
 [ 0  9  3]
 [ 0  0 11]]

```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	7
Iris-versicolor	1.00	0.75	0.86	12
Iris-virginica	0.79	1.00	0.88	11
avg / total	0.92	0.90	0.90	30

```

MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
beta_2=0.999, early_stopping=False, epsilon=1e-08,
hidden_layer_sizes=(100,), learning_rate='constant',
learning_rate_init=0.001, max_iter=200, momentum=0.9,
nesterovs_momentum=True, power_t=0.5, random_state=None,
shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
verbose=False, warm_start=False)

```

Evaluation

(Accuracy of Classification Model)

Assessing the Classification Model

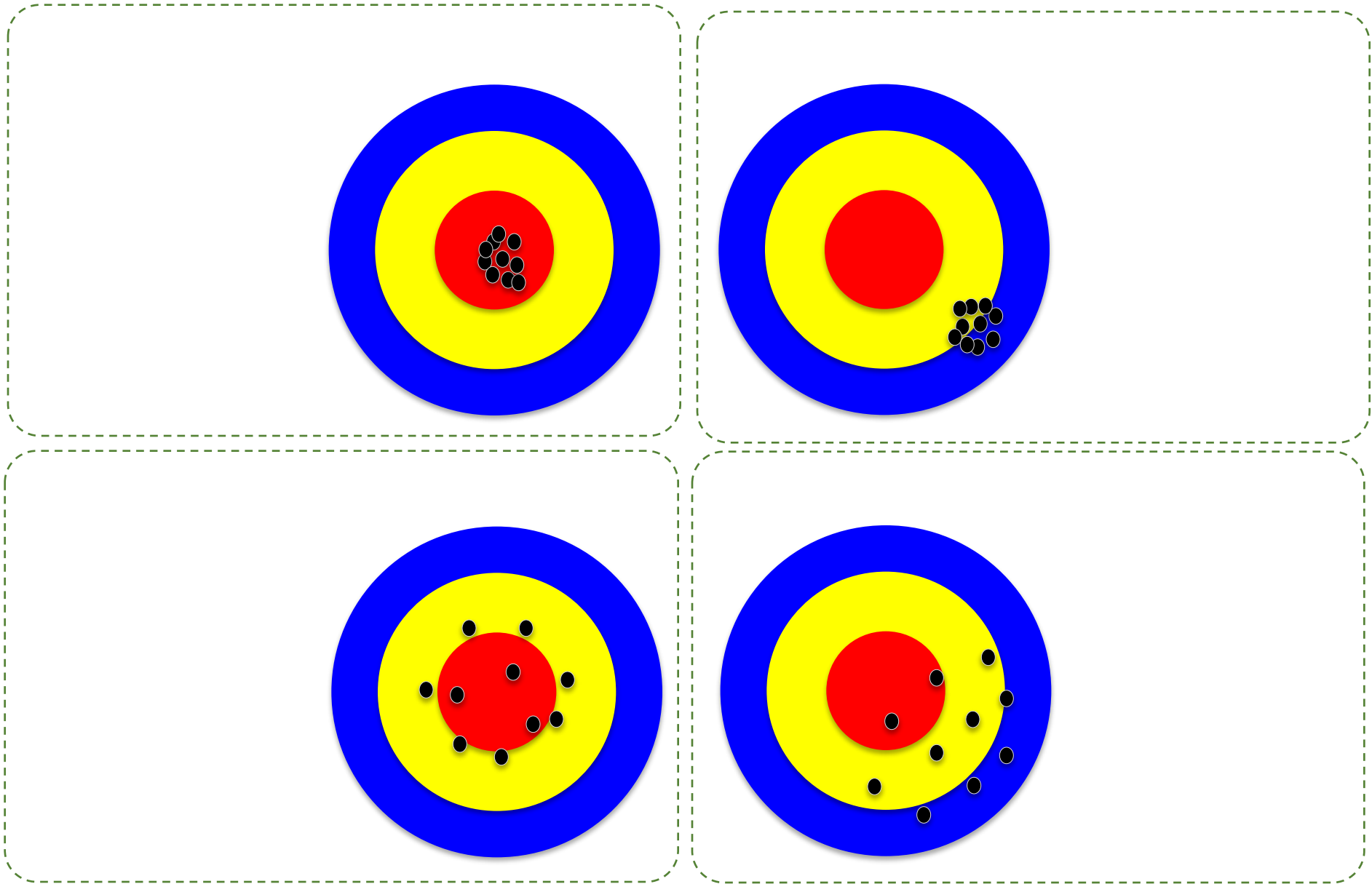
- **Predictive accuracy**
 - **Hit rate**
- **Speed**
 - **Model building; predicting**
- **Robustness**
- **Scalability**
- **Interpretability**
 - **Transparency, explainability**

Accuracy

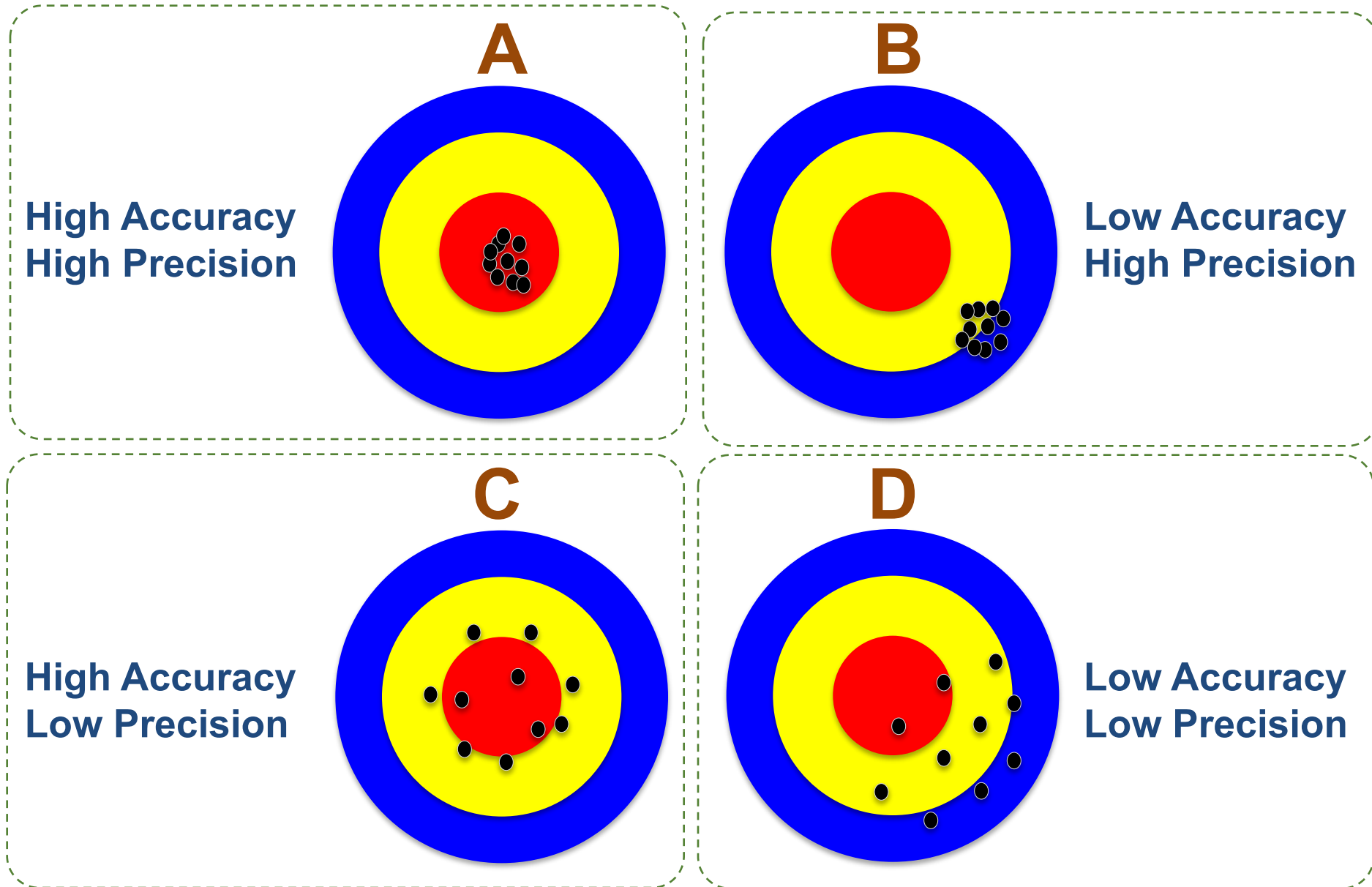
Validity

Precision

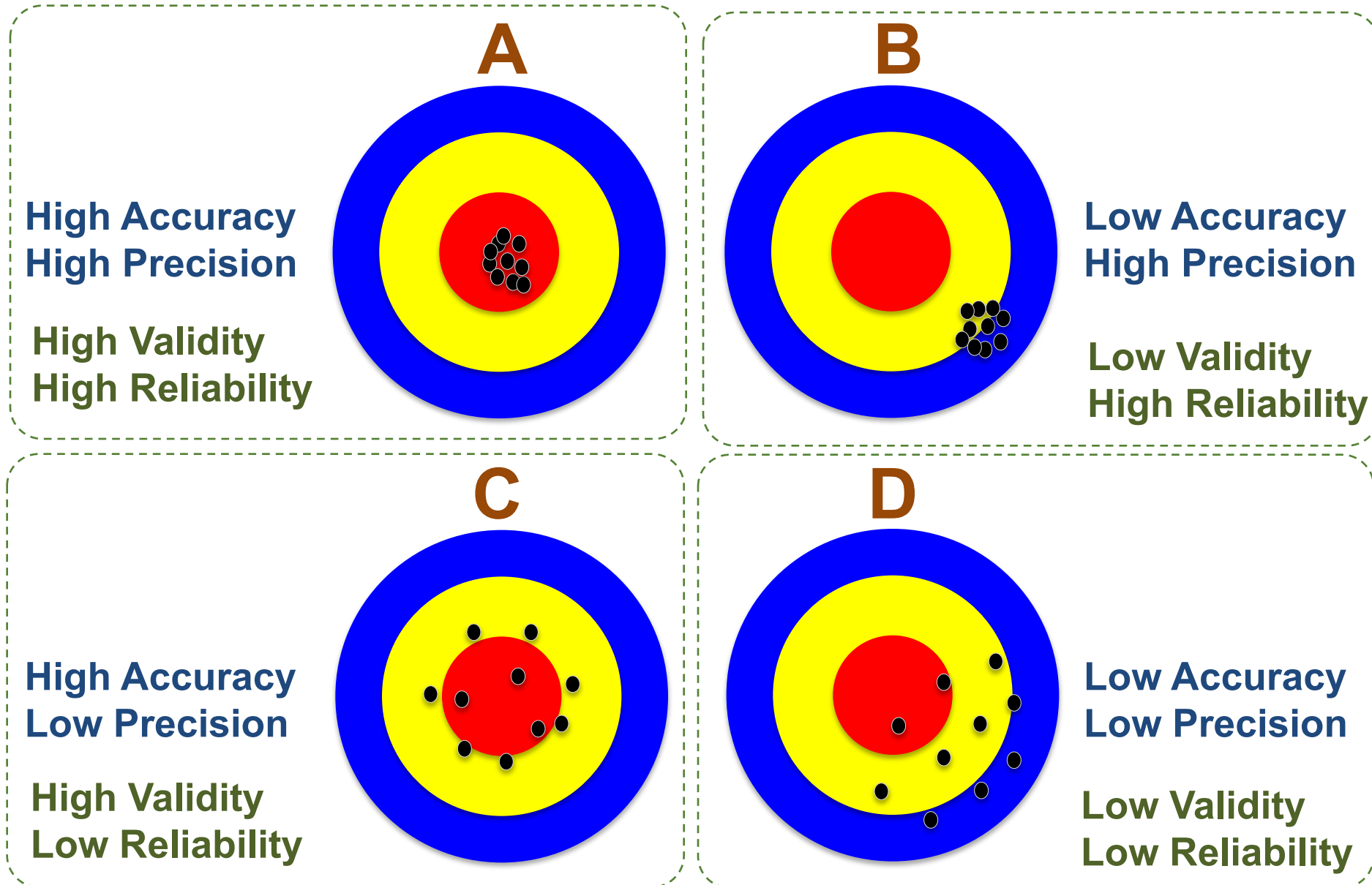
Reliability



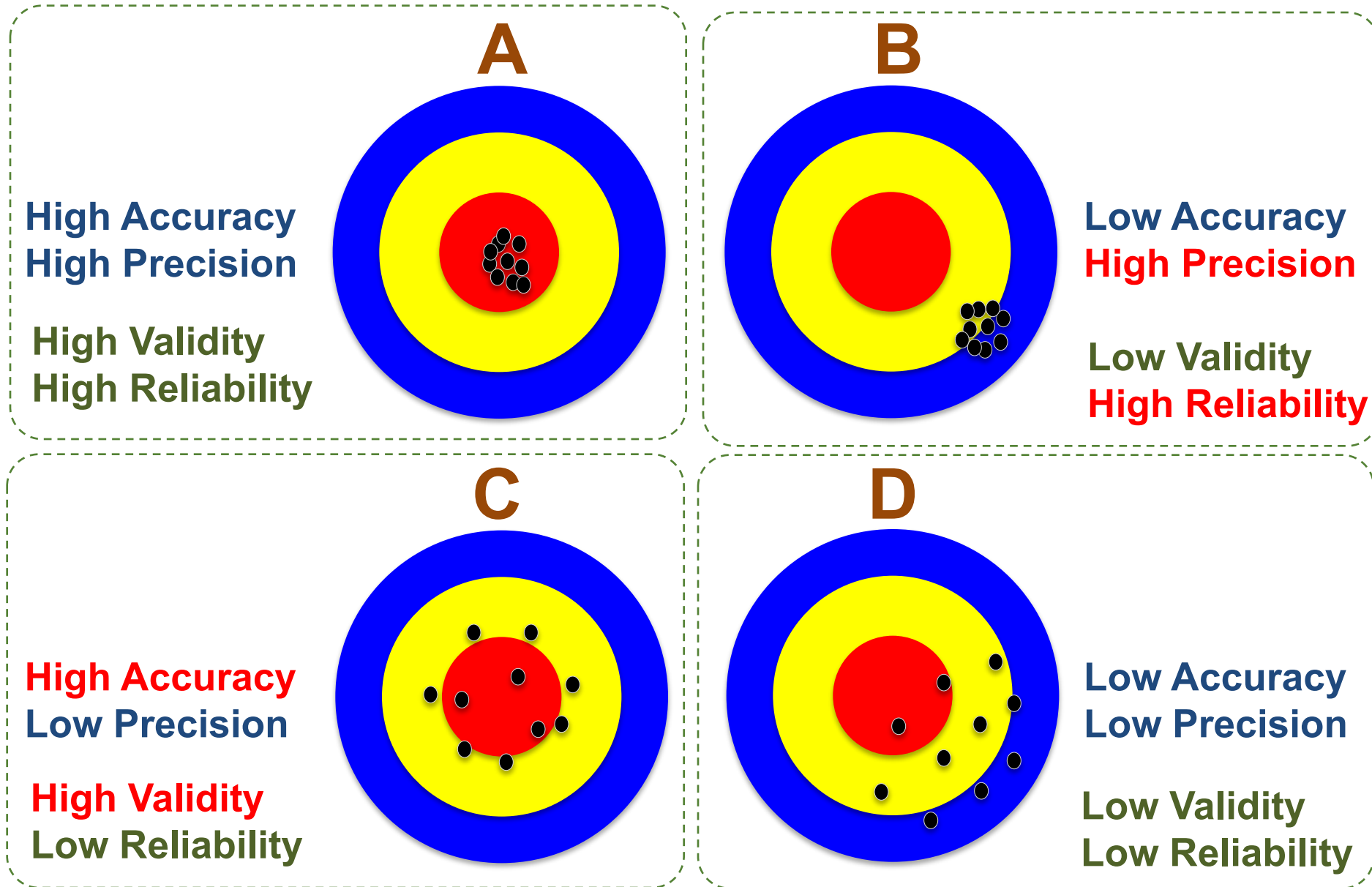
Accuracy vs. Precision



Accuracy vs. Precision



Accuracy vs. Precision



Confusion Matrix

for Tabulation of Two-Class Classification Results

		True/Observed Class	
		Positive	Negative
Predicted Class	Positive	True Positive Count (TP)	False Positive Count (FP)
	Negative	False Negative Count (FN)	True Negative Count (TN)

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$True\ Positive\ Rate = \frac{TP}{TP + FN}$$

$$True\ Negative\ Rate = \frac{TN}{TN + FP}$$

$$Precision = \frac{TP}{TP + FP}$$

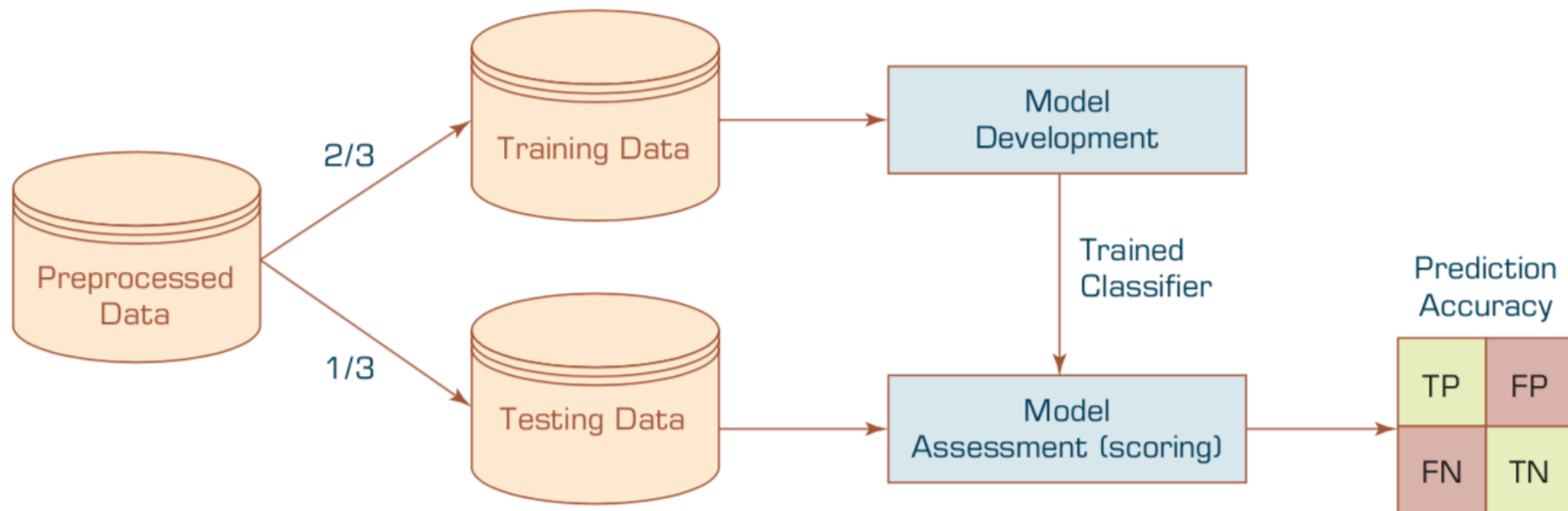
$$Recall = \frac{TP}{TP + FN}$$

Sensitivity = True Positive Rate

Specificity = True Negative Rate

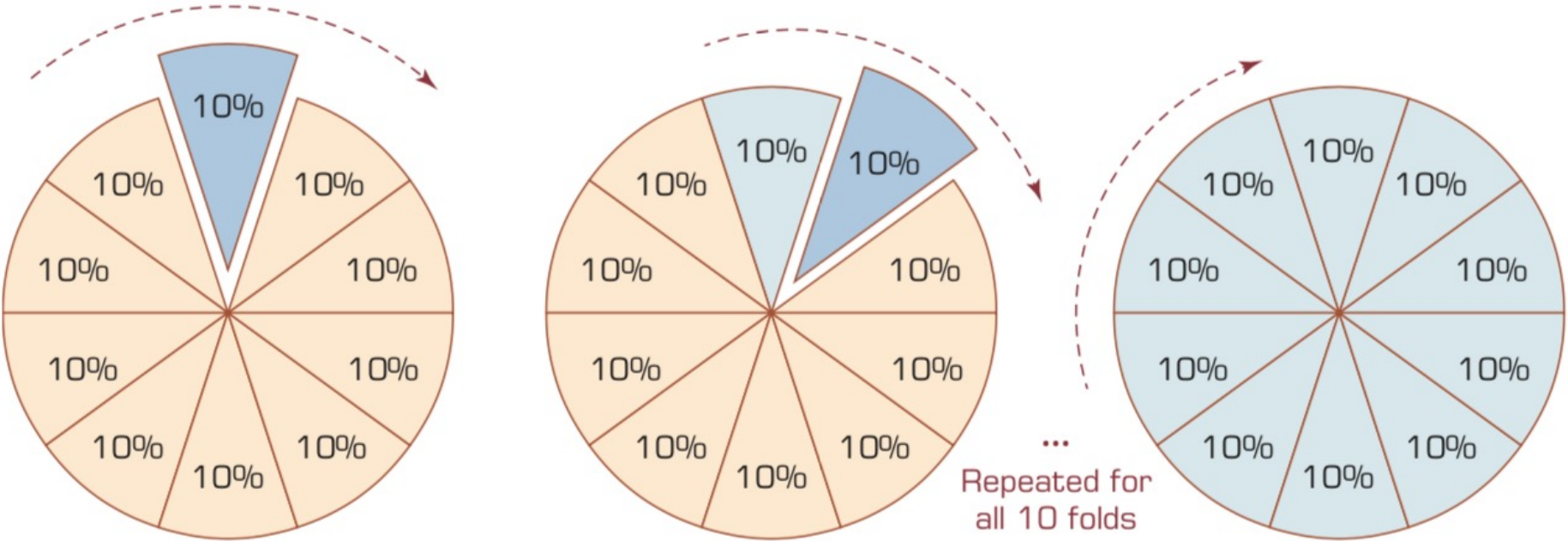
Estimation Methodologies for Classification

- **Simple split** (or holdout or test sample estimation)
 - Split the data into 2 mutually exclusive sets training (~70%) and testing (30%)



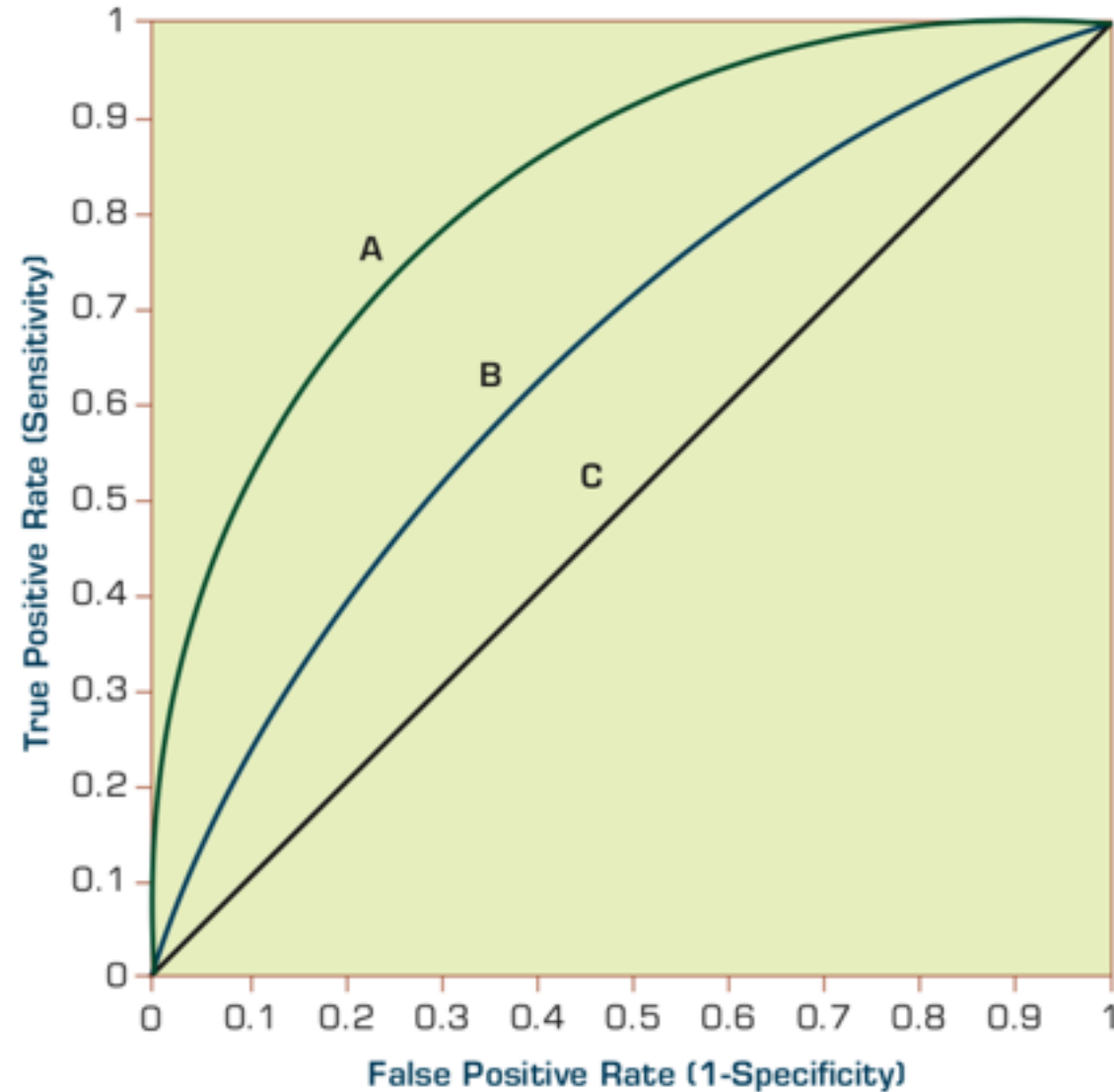
- For ANN, the data is split into three sub-sets (training [~60%], validation [~20%], testing [~20%])

k-Fold Cross-Validation



Estimation Methodologies for Classification

Area under the ROC curve



		True Class (actual value)		total
		Positive	Negative	
Predictive Class (prediction outcome)	Positive	True Positive (TP)	False Positive (FP)	P'
	Negative	False Negative (FN)	True Negative (TN)	N'
total		P	N	

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$True\ Positive\ Rate = \frac{TP}{TP + FN}$$

$$True\ Negative\ Rate = \frac{TN}{TN + FP}$$

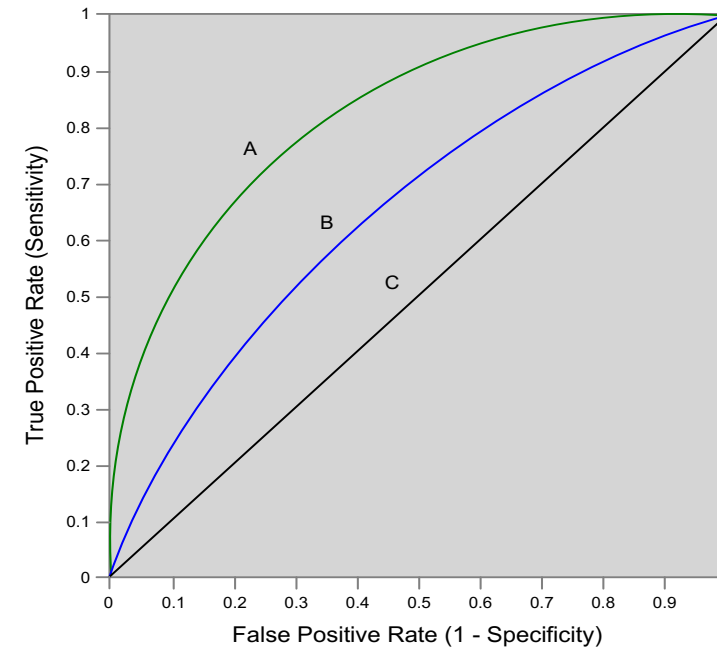
$$Precision = \frac{TP}{TP + FP} \quad Recall = \frac{TP}{TP + FN}$$

$$True\ Positive\ Rate\ (Sensitivity) = \frac{TP}{TP + FN}$$

$$True\ Negative\ Rate\ (Specificity) = \frac{TN}{TN + FP}$$

$$False\ Positive\ Rate = \frac{FP}{FP + TN}$$

$$False\ Positive\ Rate\ (1 - Specificity) = \frac{FP}{FP + TN}$$



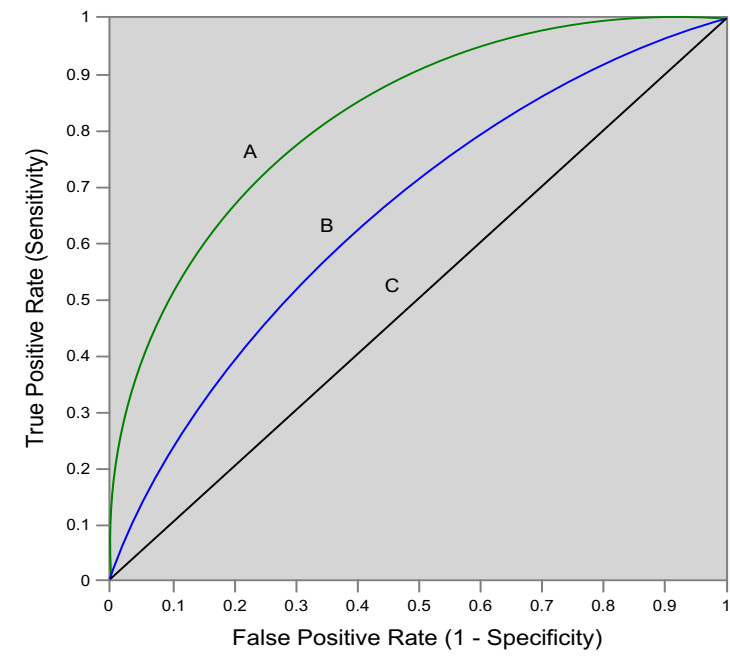
		True Class (actual value)		total
		Positive	Negative	
Predictive Class (prediction outcome)	Positive	True Positive (TP)	False Positive (FP)	P'
	Negative	False Negative (FN)	True Negative (TN)	N'
total		P	N	

$$\text{True Positive Rate} = \frac{TP}{TP + FN}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{True Positive Rate (Sensitivity)} = \frac{TP}{TP + FN}$$

Sensitivity
 = True Positive Rate
 = Recall
 = Hit rate
 = $TP / (TP + FN)$



		True Class (actual value)		total
		Positive	Negative	
Predictive Class (prediction outcome)	Positive	True Positive (TP)	False Positive (FP)	P'
	Negative	False Negative (FN)	True Negative (TN)	N'
total		P	N	

Specificity

= True Negative Rate

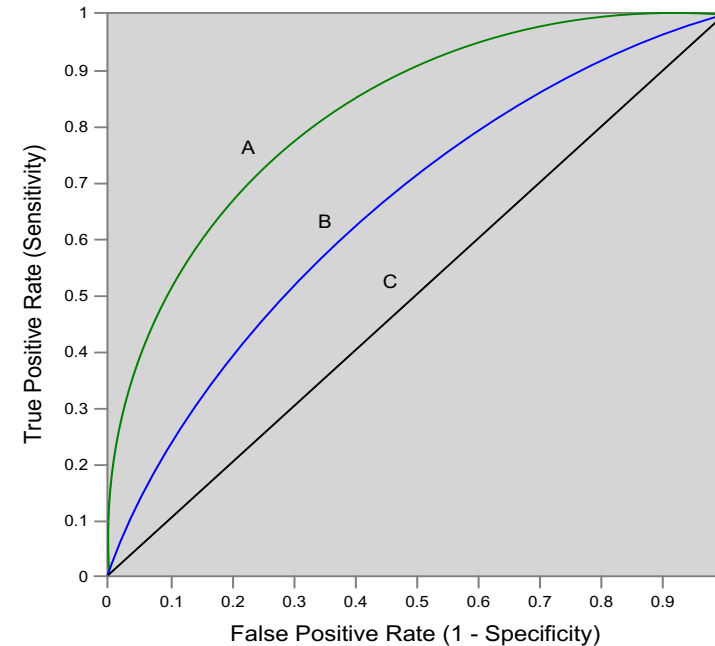
= TN / N

= $TN / (TN + FP)$

$$\text{True Negative Rate (Specificity)} = \frac{TN}{TN + FP}$$

$$\text{False Positive Rate (1-Specificity)} = \frac{FP}{FP + TN}$$

$$\text{True Negative Rate} = \frac{TN}{TN + FP}$$



		True Class (actual value)		total
		Positive	Negative	
Predictive Class (prediction outcome)	Positive	True Positive (TP)	False Positive (FP)	P'
	Negative	False Negative (FN)	True Negative (TN)	N'
total		P	N	

Precision

= Positive Predictive Value (PPV)

$$Precision = \frac{TP}{TP + FP}$$

Recall

= True Positive Rate (TPR)

= Sensitivity

= Hit Rate

$$Recall = \frac{TP}{TP + FN}$$

F1 score (F-score)(F-measure)

is the harmonic mean of precision and recall

$$= 2TP / (P + P')$$

$$= 2TP / (2TP + FP + FN)$$

$$F = 2 * \frac{precision * recall}{precision + recall}$$

A		
63 (TP)	28 (FP)	91
37 (FN)	72 (TN)	109
100	100	200

Recall

= True Positive Rate (TPR)
 = Sensitivity
 = Hit Rate
 = $TP / (TP + FN)$

Specificity

= True Negative Rate
 = TN / N
 = $TN / (TN + FP)$

$$TPR = 0.63$$

$$Recall = \frac{TP}{TP + FN}$$

$$True\ Negative\ Rate\ (Specificity) = \frac{TN}{TN + FP}$$

$$FPR = 0.28$$

$$False\ Positive\ Rate\ (1 - Specificity) = \frac{FP}{FP + TN}$$

$$PPV = 0.69$$

$$= 63 / (63 + 28)$$

$$= 63 / 91$$

$$Precision = \frac{TP}{TP + FP}$$

Precision

= Positive Predictive Value (PPV)

$$F1 = 0.66$$

$$= 2 * (0.63 * 0.69) / (0.63 + 0.69)$$

$$= (2 * 63) / (100 + 91)$$

$$= (0.63 + 0.69) / 2 = 1.32 / 2 = 0.66$$

$$F = 2 * \frac{precision * recall}{precision + recall}$$

F1 score (F-score) (F-measure)

is the harmonic mean of
 precision and recall

$$= 2TP / (P + P')$$

$$= 2TP / (2TP + FP + FN)$$

$$ACC = 0.68$$

$$= (63 + 72) / 200$$

$$= 135 / 200 = 67.5$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

A

63 (TP)	28 (FP)	91
37 (FN)	72 (TN)	109
100	100	200

$$\text{TPR} = 0.63$$

$$\text{FPR} = 0.28$$

$$\text{PPV} = 0.69$$

$$= 63 / (63 + 28)$$

$$= 63 / 91$$

$$\text{F1} = 0.66$$

$$= 2 * (0.63 * 0.69) / (0.63 + 0.69)$$

$$= (2 * 63) / (100 + 91)$$

$$= (0.63 + 0.69) / 2 = 1.32 / 2 = 0.66$$

$$\text{ACC} = 0.68$$

$$= (63 + 72) / 200$$

$$= 135 / 200 = 67.5$$

B

77 (TP)	77 (FP)	154
23 (FN)	23 (TN)	46
100	100	200

$$\text{TPR} = 0.77$$

$$\text{FPR} = 0.77$$

$$\text{PPV} = 0.50$$

$$\text{F1} = 0.61$$

$$\text{ACC} = 0.50$$

Recall

= True Positive Rate (TPR)

= Sensitivity

= Hit Rate

$$\text{Recall} = \frac{TP}{TP + FN}$$

Precision

= Positive Predictive Value (PPV)

$$\text{Precision} = \frac{TP}{TP + FP}$$

C

24 (TP)	88 (FP)	112
76 (FN)	12 (TN)	88
100	100	200

$$\text{TPR} = 0.24$$

$$\text{FPR} = 0.88$$

$$\text{PPV} = 0.21$$

$$\text{F1} = 0.22$$

$$\text{ACC} = 0.18$$

C'

76 (TP)	12 (FP)	88
24 (FN)	88 (TN)	112
100	100	200

$$\text{TPR} = 0.76$$

$$\text{FPR} = 0.12$$

$$\text{PPV} = 0.86$$

$$\text{F1} = 0.81$$

$$\text{ACC} = 0.82$$

Recall

= True Positive Rate (TPR)

= Sensitivity

= Hit Rate

$$\text{Recall} = \frac{TP}{TP + FN}$$

Precision

= Positive Predictive Value (PPV)

$$\text{Precision} = \frac{TP}{TP + FP}$$

**Machine Learning:
Unsupervised Learning:
Cluster Analysis,
Market Segmentation**

Machine Learning: Data Mining Tasks & Methods

**Unsupervised Learning:
Cluster Analysis,
Market Segmentation**

Segmentation

Data Mining Tasks & Methods	Data Mining Algorithms	Learning Type
Prediction		
Classification	Decision Trees, Neural Networks, Support Vector Machines, kNN, Naïve Bayes, GA	Supervised
Regression	Linear/Nonlinear Regression, ANN, Regression Trees, SVM, kNN, GA	Supervised
Time series	Autoregressive Methods, Averaging Methods, Exponential Smoothing, ARIMA	Supervised
Association		
Market-basket	Apriori, OneR, ZeroR, Eclat, GA	Unsupervised
Link analysis	Expectation Maximization, Apriori Algorithm, Graph-Based Matching	Unsupervised
Sequence analysis	Apriori Algorithm, FP-Growth, Graph-Based Matching	Unsupervised
Segmentation		
Clustering	k-means, Expectation Maximization (EM)	Unsupervised
Outlier analysis	k-means, Expectation Maximization (EM)	Unsupervised

Example of Cluster Analysis

Point	P	P(x,y)
p01	a	(3, 4)
p02	b	(3, 6)
p03	c	(3, 8)
p04	d	(4, 5)
p05	e	(4, 7)
p06	f	(5, 1)
p07	g	(5, 5)
p08	h	(7, 3)
p09	i	(7, 5)
p10	j	(8, 5)

K-Means Clustering

Point	P	P(x,y)	m1 distance	m2 distance	Cluster
p01	a	(3, 4)	1.95	3.78	Cluster1
p02	b	(3, 6)	0.69	4.51	Cluster1
p03	c	(3, 8)	2.27	5.86	Cluster1
p04	d	(4, 5)	0.89	3.13	Cluster1
p05	e	(4, 7)	1.22	4.45	Cluster1
p06	f	(5, 1)	5.01	3.05	Cluster2
p07	g	(5, 5)	1.57	2.30	Cluster1
p08	h	(7, 3)	4.37	0.56	Cluster2
p09	i	(7, 5)	3.43	1.52	Cluster2
p10	j	(8, 5)	4.41	1.95	Cluster2

m1 (3.67, 5.83)

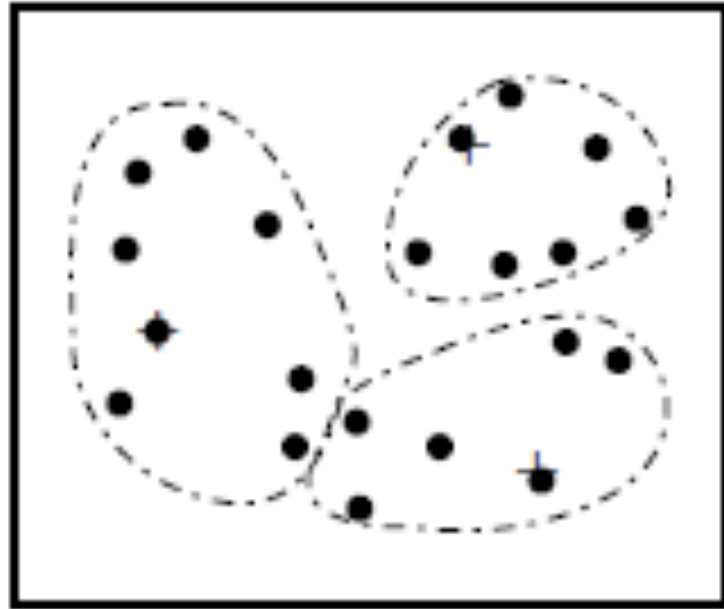
m2 (6.75, 3.50)

Cluster Analysis

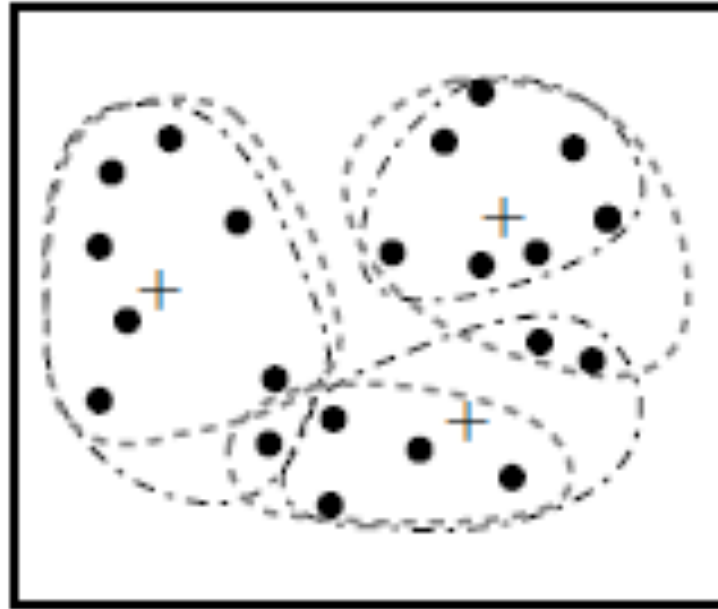
Cluster Analysis

- Used for automatic identification of **natural groupings** of things
- Part of the machine-learning family
- Employ **unsupervised learning**
- Learns the clusters of things from past data, then assigns new instances
- There is not an output variable
- Also known as **segmentation**

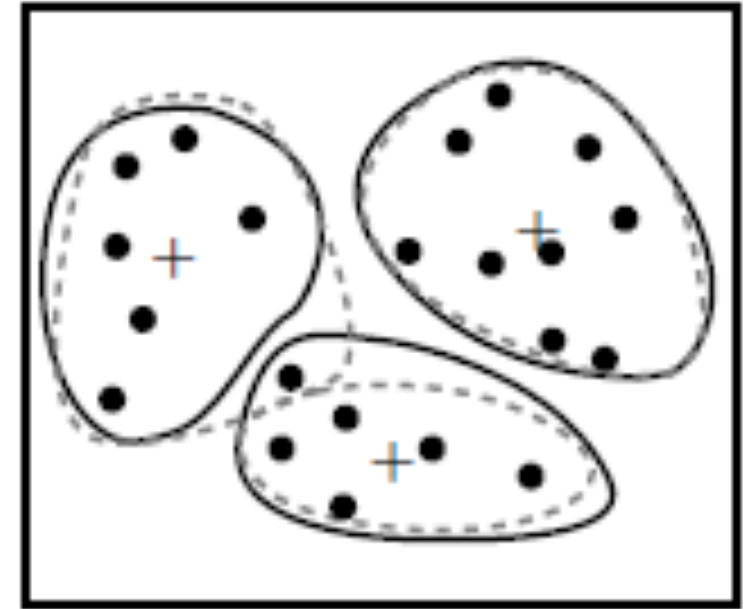
Cluster Analysis



(a)



(b)



(c)

Clustering of a set of objects based on the *k-means method*.
(The mean of each cluster is marked by a “+”.)

Cluster Analysis

- **Clustering results may be used to**
 - Identify natural **groupings of customers**
 - Identify rules for assigning new cases to classes for targeting/diagnostic purposes
 - Provide characterization, definition, labeling of populations
 - Decrease the size and complexity of problems for other data mining methods
 - Identify **outliers** in a specific domain (e.g., rare-event detection)

***k*-Means Clustering Algorithm**

- ***k*** : pre-determined number of clusters
- Algorithm (**Step 0**: determine value of ***k***)

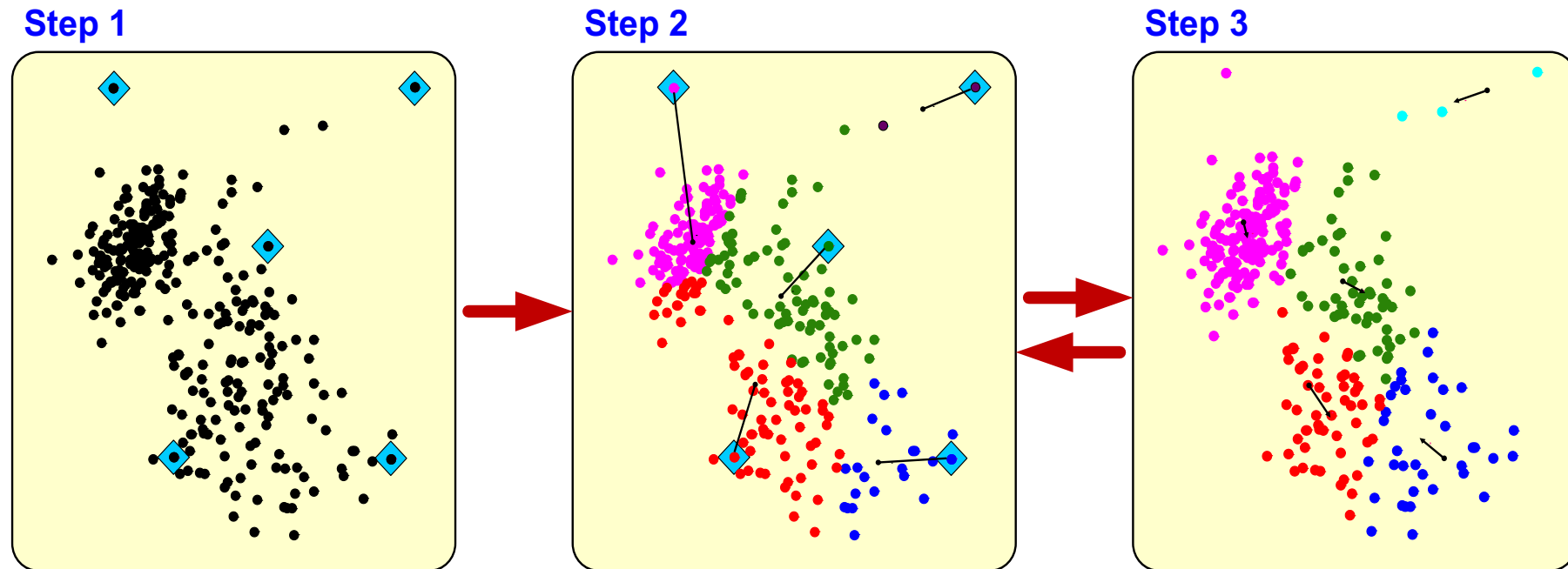
Step 1: Randomly generate ***k*** random points as initial cluster centers

Step 2: Assign each point to the nearest cluster center

Step 3: Re-compute the new cluster centers

Repetition step: Repeat steps 2 and 3 until some convergence criterion is met (usually that the assignment of points to clusters becomes stable)

Cluster Analysis for Data Mining - *k*-Means Clustering Algorithm



Similarity

Distance

Similarity and Dissimilarity Between Objects

- **Distances** are normally used to measure the **similarity** or **dissimilarity** between two data objects

- Some popular ones include: ***Minkowski distance***:

$$d(i, j) = \sqrt[q]{(|x_{i1} - x_{j1}|^q + |x_{i2} - x_{j2}|^q + \dots + |x_{ip} - x_{jp}|^q)}$$

where $i = (x_{i1}, x_{i2}, \dots, x_{ip})$ and $j = (x_{j1}, x_{j2}, \dots, x_{jp})$ are two p -dimensional data objects, and q is a positive integer

- If $q = 1$, d is **Manhattan distance**

$$d(i, j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \dots + |x_{ip} - x_{jp}|$$

Similarity and Dissimilarity Between Objects (Cont.)

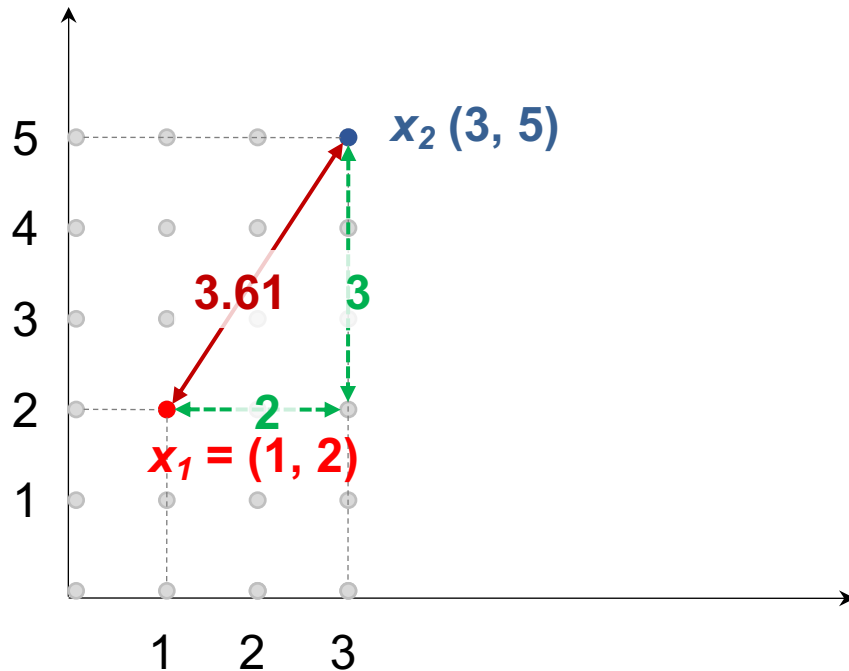
- **If $q = 2$, d is **Euclidean distance**:**

$$d(i,j) = \sqrt{(|x_{i_1} - x_{j_1}|^2 + |x_{i_2} - x_{j_2}|^2 + \dots + |x_{i_p} - x_{j_p}|^2)}$$

- **Properties**
 - $d(i,j) \geq 0$
 - $d(i,i) = 0$
 - $d(i,j) = d(j,i)$
 - $d(i,j) \leq d(i,k) + d(k,j)$
- **Also, one can use weighted distance, parametric Pearson product moment correlation, or other dissimilarity measures**

Euclidean distance vs Manhattan distance

- Distance of two point $x_1 = (1, 2)$ and $x_2 (3, 5)$

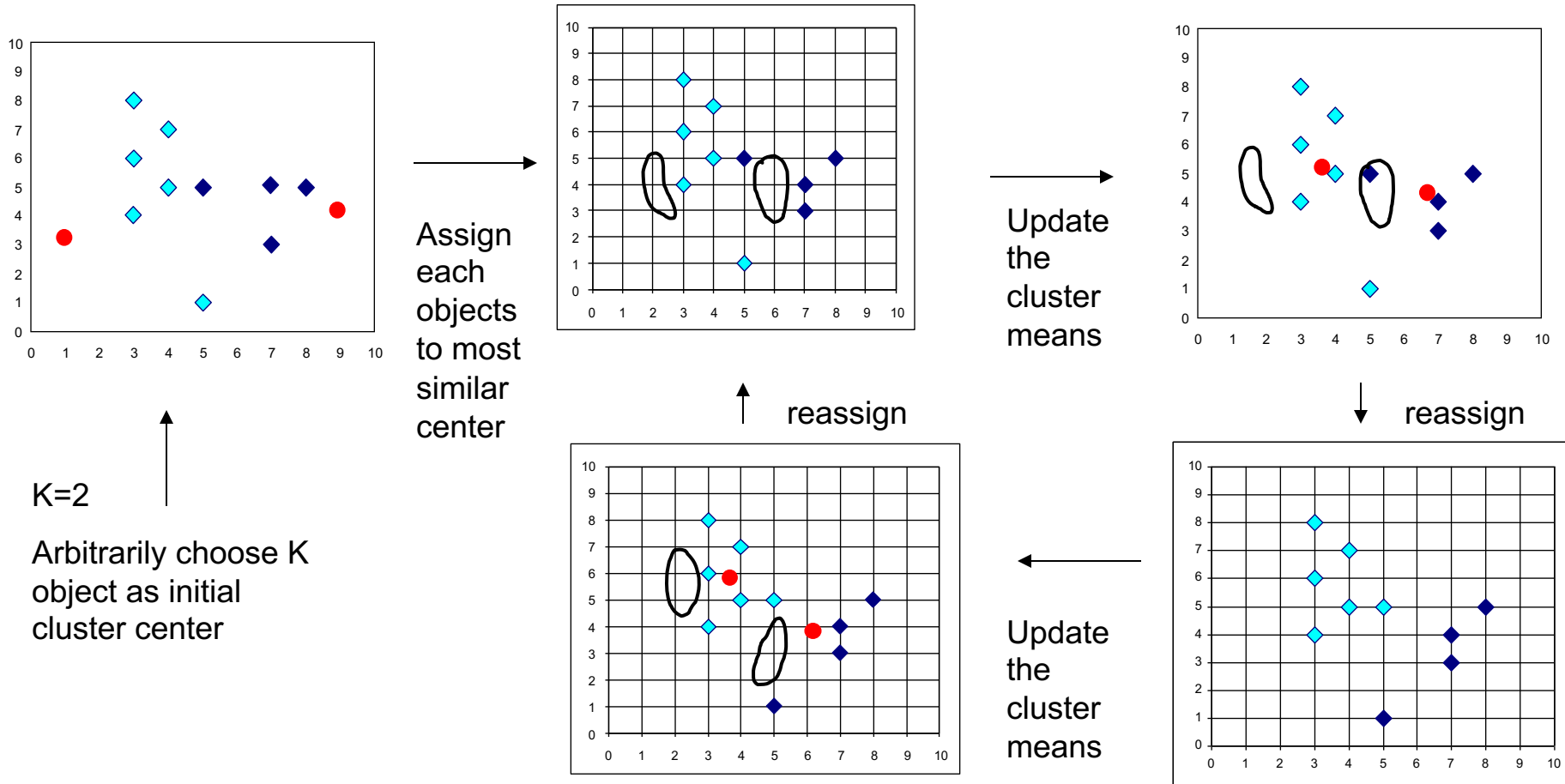


$$\begin{aligned} \text{Euclidean distance:} \\ &= ((3-1)^2 + (5-2)^2)^{1/2} \\ &= (2^2 + 3^2)^{1/2} \\ &= (4 + 9)^{1/2} \\ &= (13)^{1/2} \\ &= 3.61 \end{aligned}$$

$$\begin{aligned} \text{Manhattan distance:} \\ &= (3-1) + (5-2) \\ &= 2 + 3 \\ &= 5 \end{aligned}$$

The *K-Means* Clustering Method

- **Example**



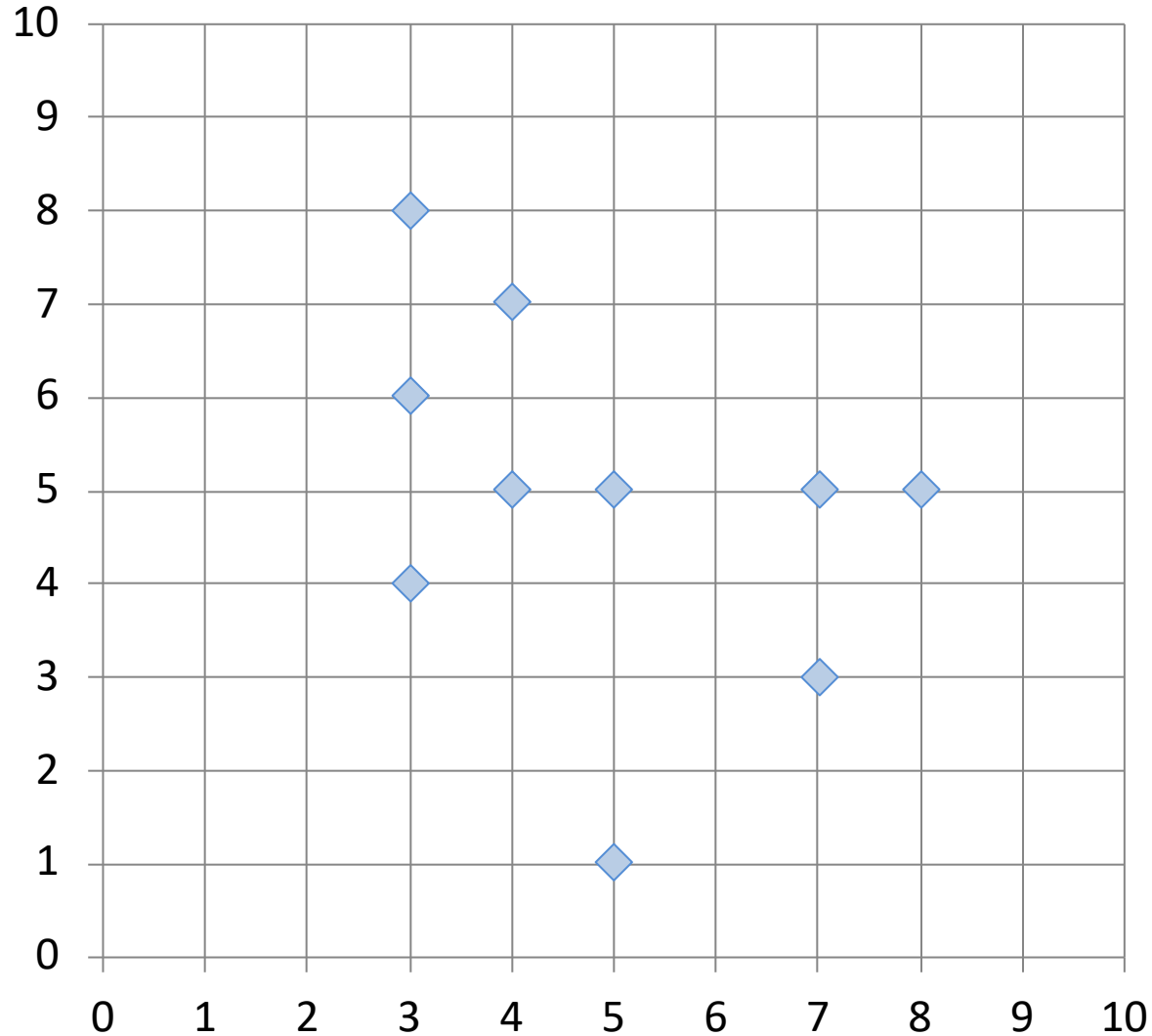
K-Means Clustering

Example of Cluster Analysis

Point	P	P(x,y)
p01	a	(3, 4)
p02	b	(3, 6)
p03	c	(3, 8)
p04	d	(4, 5)
p05	e	(4, 7)
p06	f	(5, 1)
p07	g	(5, 5)
p08	h	(7, 3)
p09	i	(7, 5)
p10	j	(8, 5)

K-Means Clustering

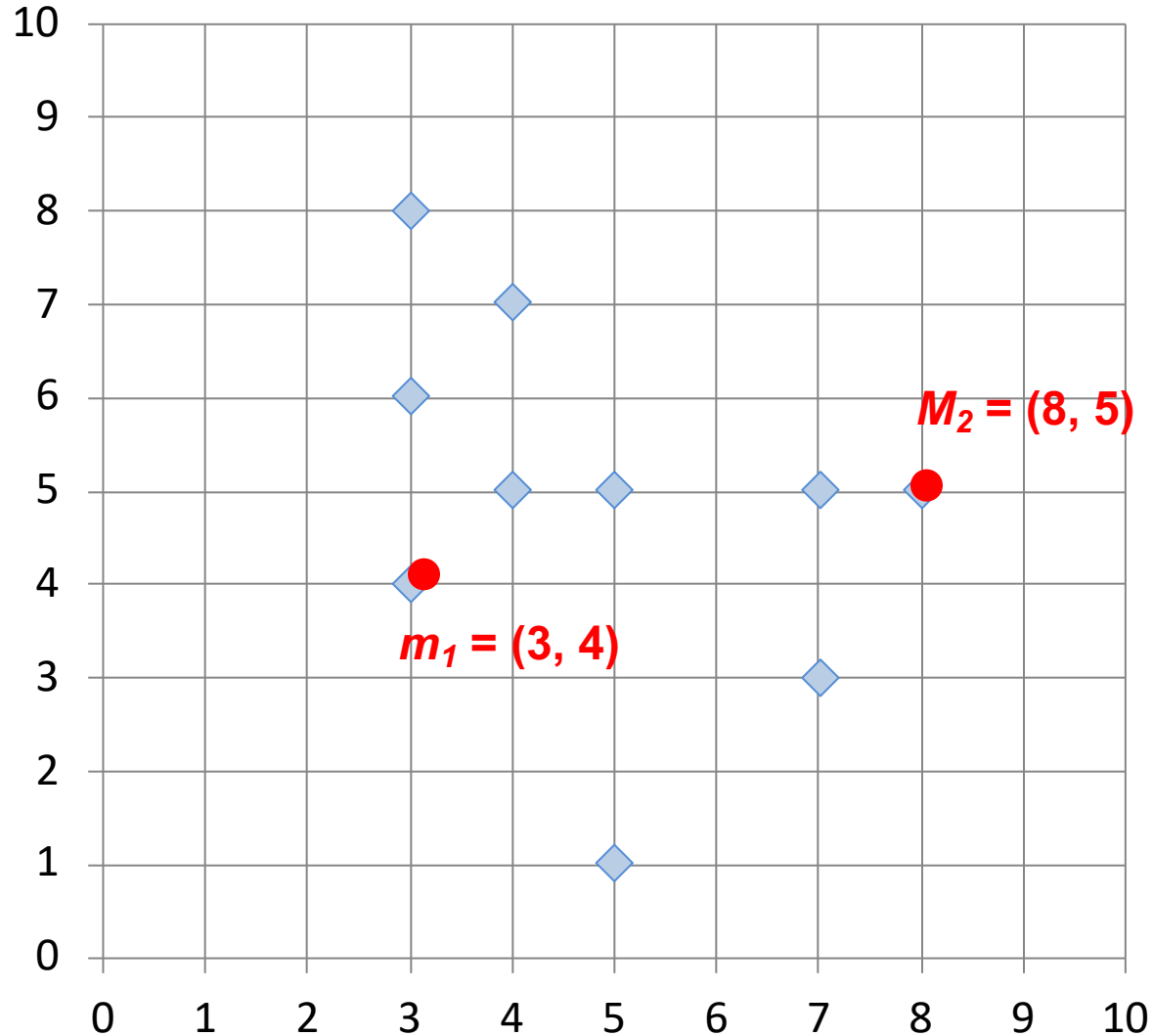
Step by Step



Point	P	P(x,y)
p01	a	(3, 4)
p02	b	(3, 6)
p03	c	(3, 8)
p04	d	(4, 5)
p05	e	(4, 7)
p06	f	(5, 1)
p07	g	(5, 5)
p08	h	(7, 3)
p09	i	(7, 5)
p10	j	(8, 5)

K-Means Clustering

Step 1: K=2, Arbitrarily choose K object as initial cluster center

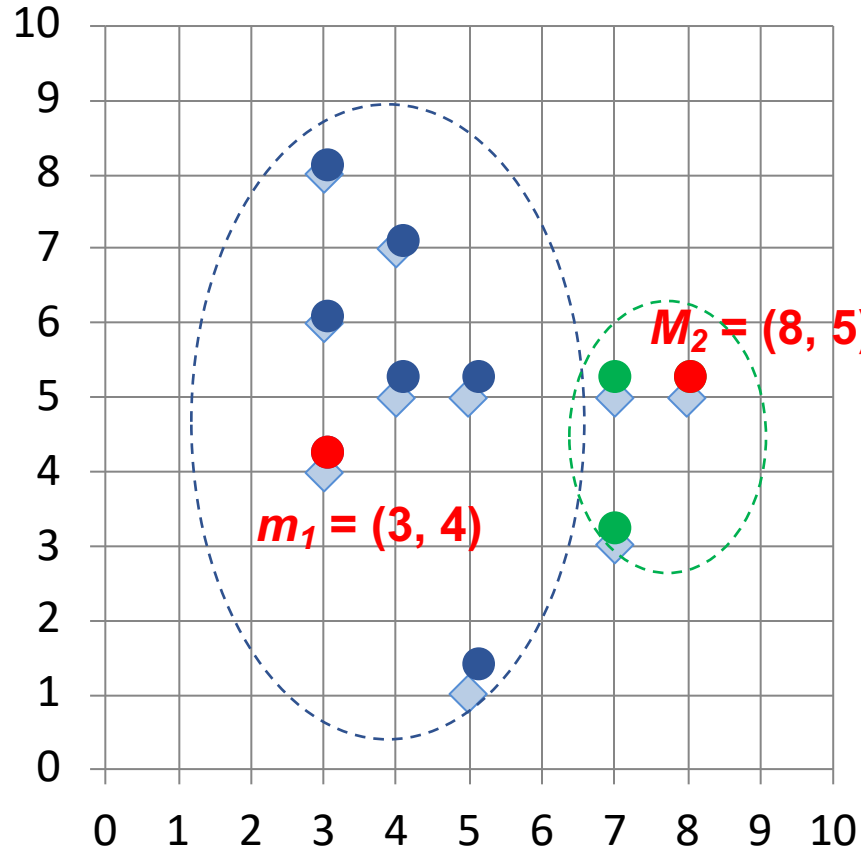


Point	P	P(x,y)
p01	a	(3, 4)
p02	b	(3, 6)
p03	c	(3, 8)
p04	d	(4, 5)
p05	e	(4, 7)
p06	f	(5, 1)
p07	g	(5, 5)
p08	h	(7, 3)
p09	i	(7, 5)
p10	j	(8, 5)

Initial m_1 (3, 4)
Initial m_2 (8, 5)

Step 2: Compute seed points as the centroids of the clusters of the current partition

Step 3: Assign each objects to most similar center



Point	P	P(x,y)	m1 distance	m2 distance	Cluster
p01	a	(3, 4)	0.00	5.10	Cluster1
p02	b	(3, 6)	2.00	5.10	Cluster1
p03	c	(3, 8)	4.00	5.83	Cluster1
p04	d	(4, 5)	1.41	4.00	Cluster1
p05	e	(4, 7)	3.16	4.47	Cluster1
p06	f	(5, 1)	3.61	5.00	Cluster1
p07	g	(5, 5)	2.24	3.00	Cluster1
p08	h	(7, 3)	4.12	2.24	Cluster2
p09	i	(7, 5)	4.12	1.00	Cluster2
p10	j	(8, 5)	5.10	0.00	Cluster2

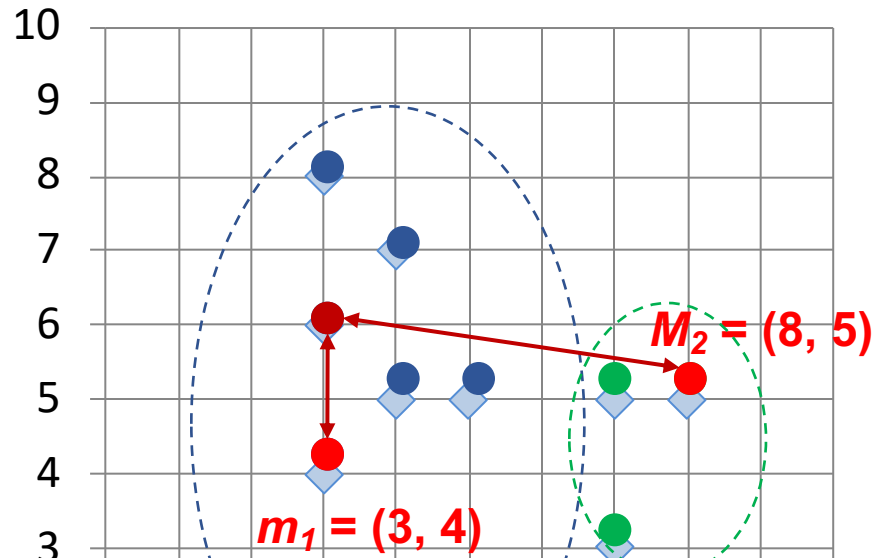
Initial m1 (3, 4)

Initial m2 (8, 5)

***K-Means* Clustering**

Step 2: Compute seed points as the centroids of the clusters of the current partition

Step 3: Assign each objects to most similar center



Euclidean distance
 $b(3,6) \leftrightarrow m1(3,4)$
 $= ((3-3)^2 + (4-6)^2)^{1/2}$
 $= (0^2 + (-2)^2)^{1/2}$
 $= (0 + 4)^{1/2}$
 $= (4)^{1/2}$
 $= 2.00$

Point	P	P(x,y)	m1 distance	m2 distance	Cluster
p01	a	(3, 4)	0.00	5.10	Cluster1
p02	b	(3, 6)	2.00	5.10	Cluster1
p03	c	(3, 8)	4.00	5.83	Cluster1
p04	d	(4, 5)	1.41	4.00	Cluster1

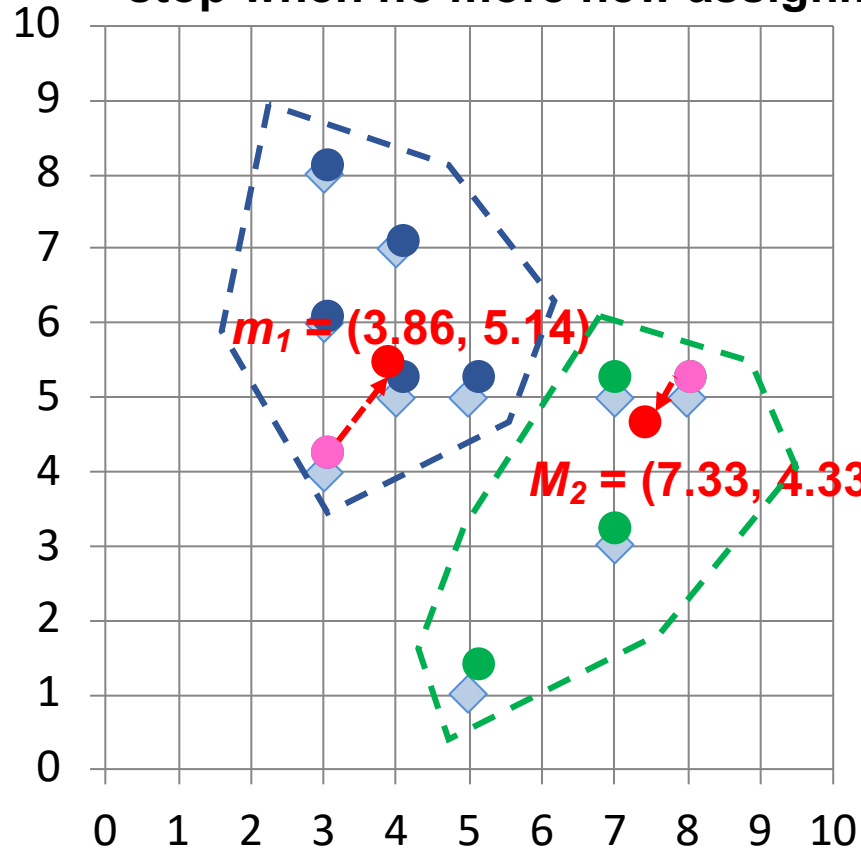
Euclidean distance
 $b(3,6) \leftrightarrow m2(8,5)$
 $= ((8-3)^2 + (5-6)^2)^{1/2}$
 $= (5^2 + (-1)^2)^{1/2}$
 $= (25 + 1)^{1/2}$
 $= (26)^{1/2}$
 $= 5.10$

Initial m1 (3, 4)
 Initial m2 (8, 5)

K-1

Cluster1
 Cluster1
 Cluster1
 Cluster1
 Cluster1
 Cluster1
 Cluster2
 Cluster2
 Cluster2

**Step 4: Update the cluster means,
Repeat Step 2, 3,
stop when no more new assignment**



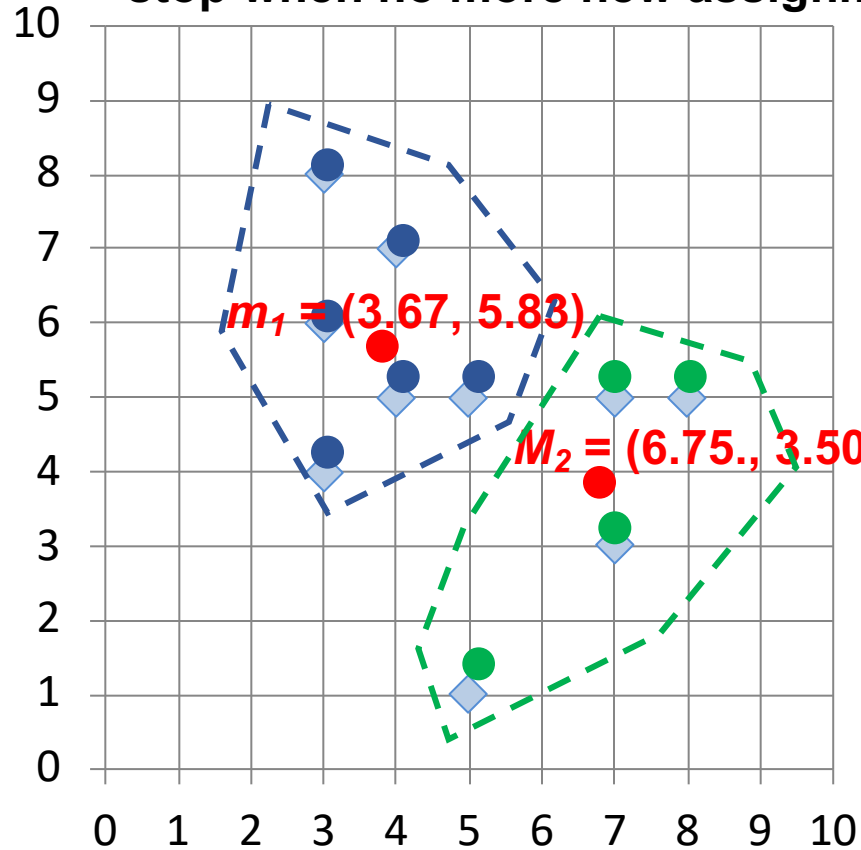
Point	P	P(x,y)	m1 distance	m2 distance	Cluster
p01	a	(3, 4)	1.43	4.34	Cluster1
p02	b	(3, 6)	1.22	4.64	Cluster1
p03	c	(3, 8)	2.99	5.68	Cluster1
p04	d	(4, 5)	0.20	3.40	Cluster1
p05	e	(4, 7)	1.87	4.27	Cluster1
p06	f	(5, 1)	4.29	4.06	Cluster2
p07	g	(5, 5)	1.15	2.42	Cluster1
p08	h	(7, 3)	3.80	1.37	Cluster2
p09	i	(7, 5)	3.14	0.75	Cluster2
p10	j	(8, 5)	4.14	0.95	Cluster2

K-Means Clustering

m1 (3.86, 5.14)

m2 (7.33, 4.33)

**Step 4: Update the cluster means,
Repeat Step 2, 3,
stop when no more new assignment**



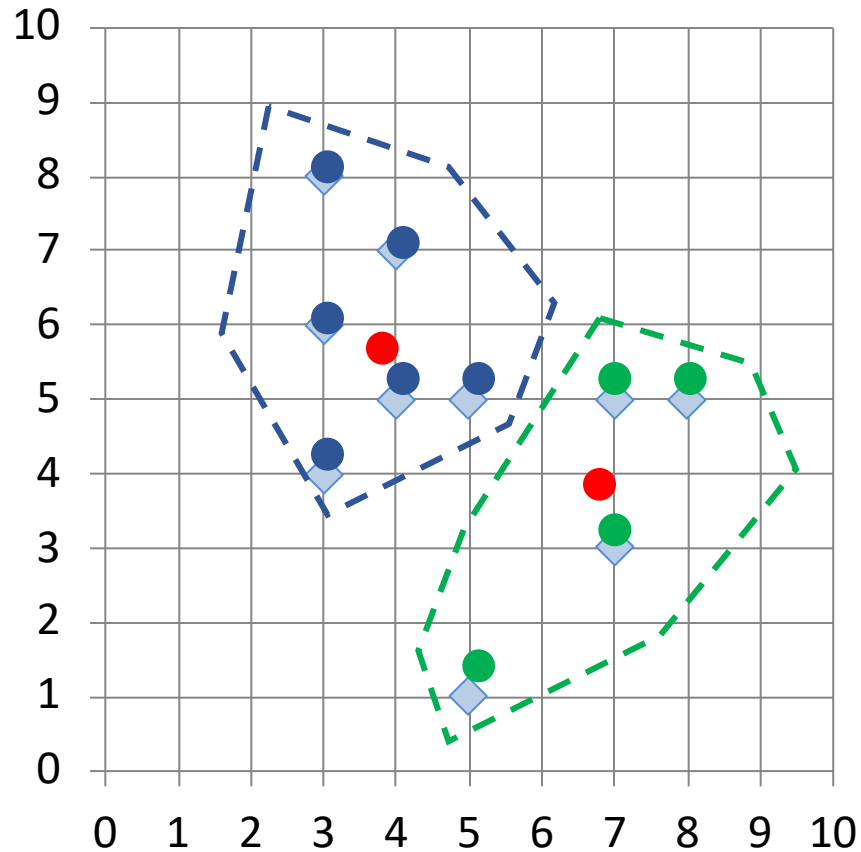
Point	P	P(x,y)	m1 distance	m2 distance	Cluster
p01	a	(3, 4)	1.95	3.78	Cluster1
p02	b	(3, 6)	0.69	4.51	Cluster1
p03	c	(3, 8)	2.27	5.86	Cluster1
p04	d	(4, 5)	0.89	3.13	Cluster1
p05	e	(4, 7)	1.22	4.45	Cluster1
p06	f	(5, 1)	5.01	3.05	Cluster2
p07	g	(5, 5)	1.57	2.30	Cluster1
p08	h	(7, 3)	4.37	0.56	Cluster2
p09	i	(7, 5)	3.43	1.52	Cluster2
p10	j	(8, 5)	4.41	1.95	Cluster2

***K-Means* Clustering**

m1 (3.67, 5.83)

m2 (6.75, 3.50)

stop when no more new assignment



Point	P	P(x,y)	m1 distance	m2 distance	Cluster
p01	a	(3, 4)	1.95	3.78	Cluster1
p02	b	(3, 6)	0.69	4.51	Cluster1
p03	c	(3, 8)	2.27	5.86	Cluster1
p04	d	(4, 5)	0.89	3.13	Cluster1
p05	e	(4, 7)	1.22	4.45	Cluster1
p06	f	(5, 1)	5.01	3.05	Cluster2
p07	g	(5, 5)	1.57	2.30	Cluster1
p08	h	(7, 3)	4.37	0.56	Cluster2
p09	i	(7, 5)	3.43	1.52	Cluster2
p10	j	(8, 5)	4.41	1.95	Cluster2

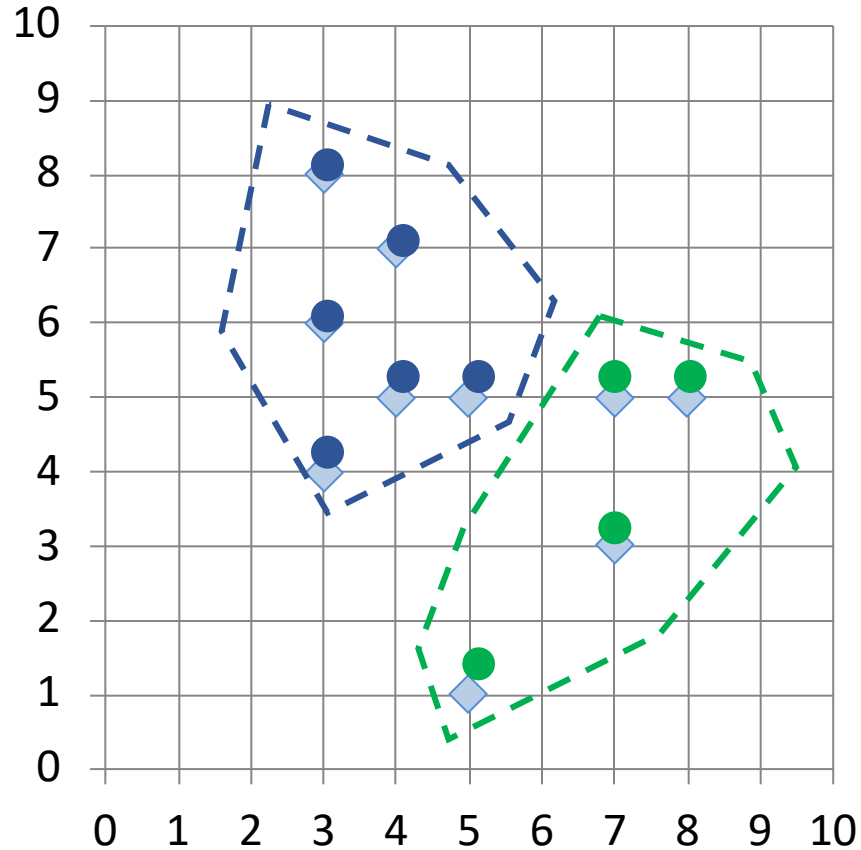
m1 (3.67, 5.83)

m2 (6.75, 3.50)

***K-Means* Clustering**

K-Means Clustering ($K=2$, two clusters)

stop when no more new assignment



Point	P	P(x,y)	m1 distance	m2 distance	Cluster
p01	a	(3, 4)	1.95	3.78	Cluster1
p02	b	(3, 6)	0.69	4.51	Cluster1
p03	c	(3, 8)	2.27	5.86	Cluster1
p04	d	(4, 5)	0.89	3.13	Cluster1
p05	e	(4, 7)	1.22	4.45	Cluster1
p06	f	(5, 1)	5.01	3.05	Cluster2
p07	g	(5, 5)	1.57	2.30	Cluster1
p08	h	(7, 3)	4.37	0.56	Cluster2
p09	i	(7, 5)	3.43	1.52	Cluster2
p10	j	(8, 5)	4.41	1.95	Cluster2

K-Means Clustering

m1 (3.67, 5.83)

m2 (6.75, 3.50)

K-Means Clustering

Point	P	P(x,y)	m1 distance	m2 distance	Cluster
p01	a	(3, 4)	1.95	3.78	Cluster1
p02	b	(3, 6)	0.69	4.51	Cluster1
p03	c	(3, 8)	2.27	5.86	Cluster1
p04	d	(4, 5)	0.89	3.13	Cluster1
p05	e	(4, 7)	1.22	4.45	Cluster1
p06	f	(5, 1)	5.01	3.05	Cluster2
p07	g	(5, 5)	1.57	2.30	Cluster1
p08	h	(7, 3)	4.37	0.56	Cluster2
p09	i	(7, 5)	3.43	1.52	Cluster2
p10	j	(8, 5)	4.41	1.95	Cluster2

m1 (3.67, 5.83)

m2 (6.75, 3.50)

Machine Learning:
Unsupervised Learning:
Cluster Analysis
K-Means Clustering

Python in Google Colab (Python101)

<https://colab.research.google.com/drive/1FEG6DnGvwfUbeo4zJ1zTunjMqf2RkCrT>

python101.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

Comment Share Settings A

RAM Disk Editing

Table of contents

- Python101
- Python File Input / Output
- OS, IO, files, and Google Drive
- Python Try Except
- Python Class
- Python Programming
- Python String and Text
- Python Numpy
- Python Pandas
- Python Data Visualization
- Unsupervised Learning:
 - Association Analysis, Market Basket Analysis
 - Association Rules Generation from Frequent Itemsets
 - Market Basket Analysis
 - Unsupervised Learning: Cluster Analysis, Market Segmentation
 - Cluster Analysis: K-Means Clustering**
 - Market Segmentation
 - Mall Customer Segmentation

```
1 import pandas as pd
2 from sklearn.cluster import KMeans
3 import plotly.express as px
4 data = {'X': [3, 3, 3, 4, 4, 5, 5, 7, 7, 8],
5         'Y': [4, 6, 8, 5, 7, 1, 5, 3, 5, 5]}
6
7 df = pd.DataFrame(data, columns=['X', 'Y'])
8 print(df)
9 kmeans = KMeans(n_clusters=2)
10 cluster = kmeans.fit_predict(df[['X', 'Y']])
11 df['Cluster'] = cluster
12 print(df)
13 px.scatter(data_frame=df, x=df['X'], y=df['Y'], color=df['cluster'], range_x = (0,10), range_y = (0,10))
```

	X	Y	Cluster
0	3	4	0
1	3	6	0
2	3	8	0
3	4	5	0
4	4	7	0
5	5	1	1
6	5	5	1
7	7	3	1
8	7	5	1
9	8	5	1

K-Means Clustering

<https://tinyurl.com/aintpupython101>

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=2)
cluster = kmeans.fit_predict(df[['X', 'Y']])
```

```
1 import pandas as pd
2 from sklearn.cluster import KMeans
3 import plotly.express as px
4 data = {'X': [3, 3, 3, 4, 4, 5, 5, 7, 7, 8],
5         'Y': [4, 6, 8, 5, 7, 1, 5, 3, 5, 5]}
6
7 df = pd.DataFrame(data, columns=['X', 'Y'])
8 print(df)
9 kmeans = KMeans(n_clusters=2)
10 cluster = kmeans.fit_predict(df[['X', 'Y']])
11 df['Cluster'] = cluster
12 print(df)
13 px.scatter(data_frame=df, x=df['X'], y=df['Y'], color=df['cluster'], range_x = (0,10), range_y = (0,10), title='K-Means Clustering')
```

```
from sklearn.cluster import KMeans  
kmeans = KMeans(n_clusters=2)  
cluster = kmeans.fit_predict(df[['X', 'Y']])
```

```
import pandas as pd  
from sklearn.cluster import KMeans  
import plotly.express as px  
data = {'X': [3, 3, 3, 4, 4, 5, 5, 7, 7, 8],  
        'Y': [4, 6, 8, 5, 7, 1, 5, 3, 5, 5]}  
df = pd.DataFrame(data, columns =['X', 'Y'])  
print(df)  
kmeans = KMeans(n_clusters=2)  
cluster = kmeans.fit_predict(df[['X', 'Y']])  
df['Cluster'] = cluster  
print(df)  
px.scatter(data_frame=df, x=df['X'], y=df['Y'],  
           color=df['cluster'], range_x = (0,10), range_y = (0,10),  
           title='K-Means Clustering')
```

K-Means Clustering

```
1 #importing the libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 %matplotlib inline
5 import pandas as pd
6
7 #importing the Iris dataset with pandas
8 # Load dataset
9 url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
10 names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
11 df = pd.read_csv(url, names=names)
12
13 array = df.values
14 X = array[:,0:4]
15 Y = array[:,4]
16
17 #Finding the optimum number of clusters for k-means classification
18 from sklearn.cluster import KMeans
19 wcss = []
20
21 for i in range(1, 8):
22     kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
23     kmeans.fit(X)
24     wcss.append(kmeans.inertia_)
25
26 #Plotting the results onto a line graph, allowing us to observe 'The elbow'
27 plt.rcParams["figure.figsize"] = (10,8)
28 plt.plot(range(1, 8), wcss)
29 plt.title('The elbow method')
30 plt.xlabel('Number of clusters')
31 plt.ylabel('WCSS') #within cluster sum of squares
32 plt.show()
```

```
#importing the libraries
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd

#importing the Iris dataset with pandas
# Load dataset
url = "https://archive.ics.uci.edu/ml/machine-
learning-databases/iris/iris.data"
names = ['sepal-length', 'sepal-width',
'petal-length', 'petal-width', 'class']
df = pd.read_csv(url, names=names)

array = df.values
X = array[:,0:4]
Y = array[:,4]
```

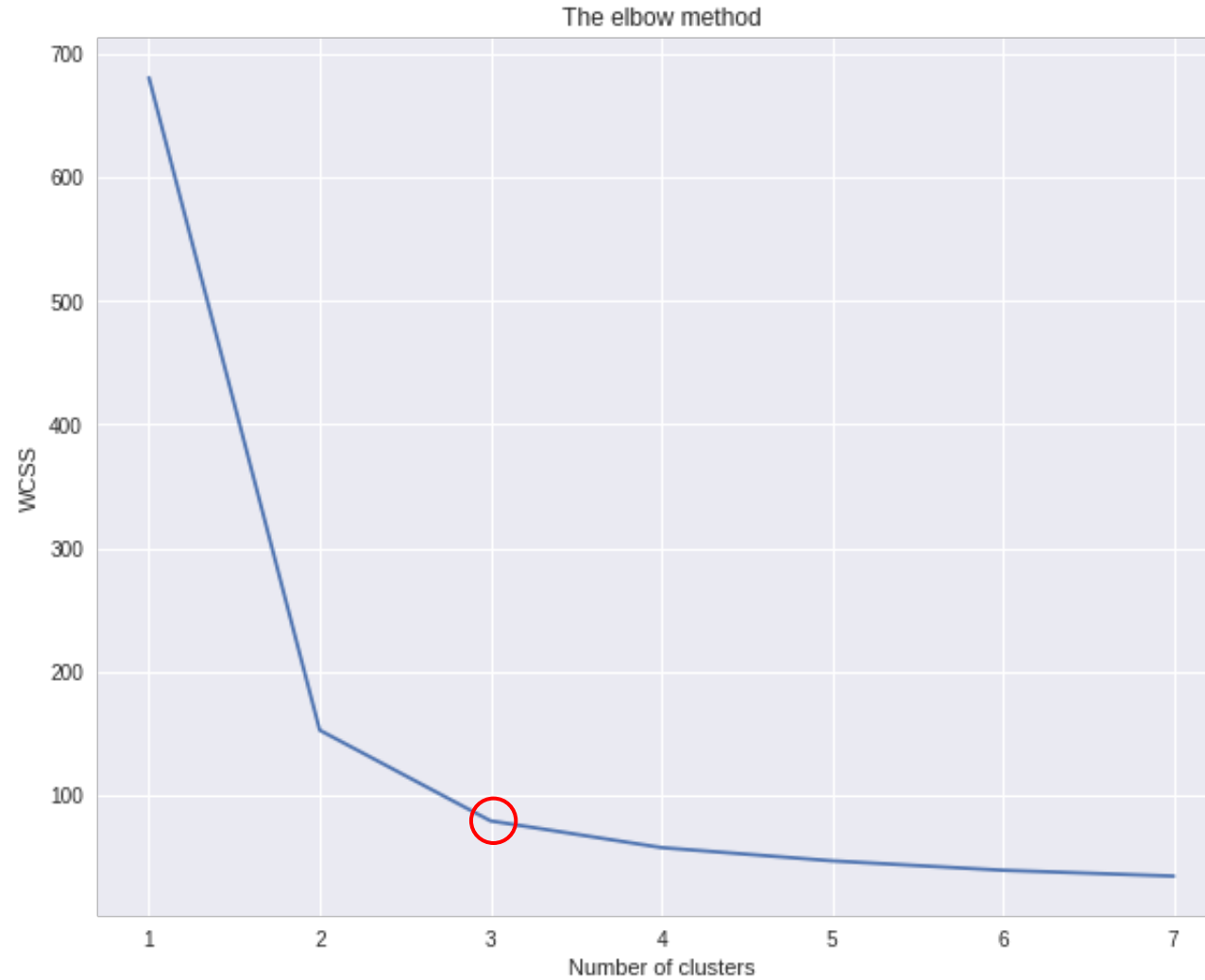
```
#Finding the optimum number of clusters for k-means
classification
from sklearn.cluster import KMeans
wcss = []

for i in range(1, 8):
    kmeans = KMeans(n_clusters = i, init = 'k-means++',
max_iter = 300, n_init = 10, random_state = 0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

#Plotting the results onto a line graph, allowing us to
observe 'The elbow'
plt.rcParams["figure.figsize"] = (10,8)
plt.plot(range(1, 8), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS') #within cluster sum of squares
plt.show()
```

K-Means Clustering

The elbow method ($k=3$)




```
kmeans = KMeans(n_clusters = 3,  
init = 'k-means++', max_iter = 300,  
n_init = 10, random_state = 0)  
y_kmeans = kmeans.fit_predict(X)
```

```
1 #Applying kmeans to the dataset / Creating the kmeans classifier  
2 kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)  
3 y_kmeans = kmeans.fit_predict(X).
```

```
#Visualising the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100,
c = 'red', label = 'Iris-setosa')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100,
c = 'blue', label = 'Iris-versicolour')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100,
c = 'green', label = 'Iris-virginica')

#Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0],
kmeans.cluster_centers_[:,1], s = 100, c = 'yellow', label =
'Centroids')

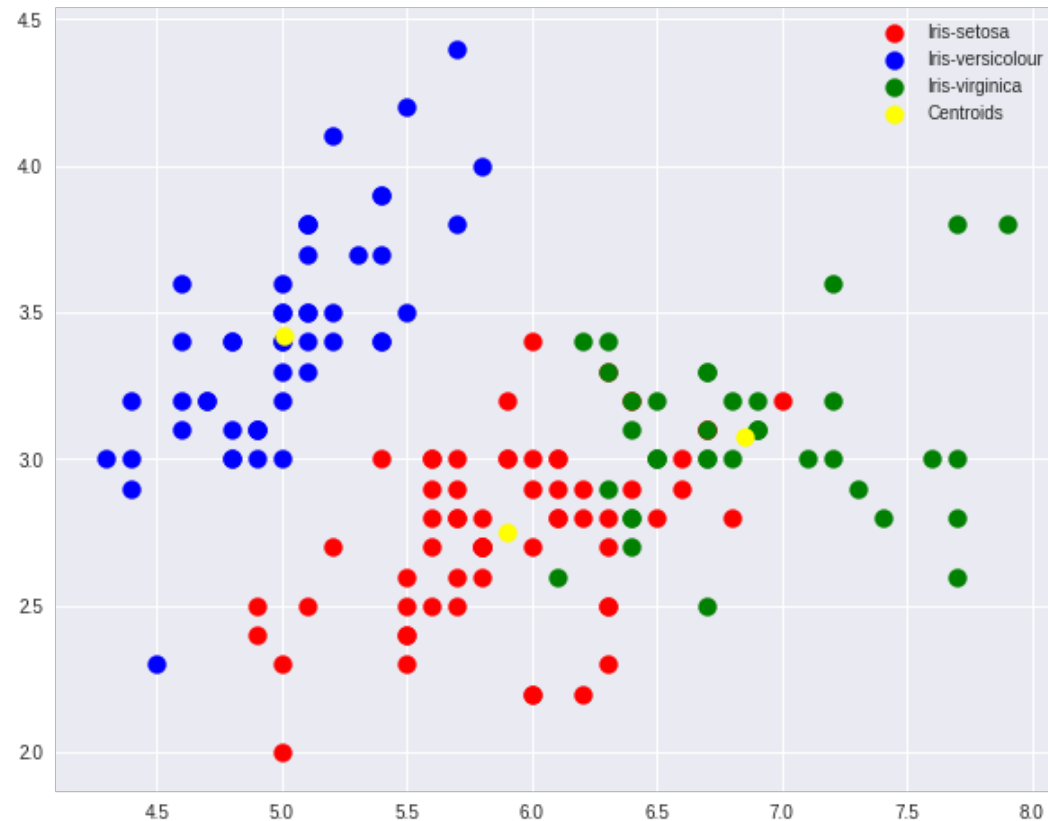
plt.legend()
```

```
1 #Visualising the clusters
2 plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Iris-setosa')
3 plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Iris-versicolour')
4 plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Iris-virginica')
5
6 #Plotting the centroids of the clusters
7 plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 100, c = 'yellow', label = 'Centroids')
8
9 plt.legend()
```

K-Means Clustering

```
1 #Applying kmeans to the dataset / Creating the kmeans classifier
2 kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
3 y_kmeans = kmeans.fit_predict(X).
```

```
1 #Visualising the clusters
2 plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Iris-setosa')
3 plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Iris-versicolour')
4 plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Iris-virginica')
5
6 #Plotting the centroids of the clusters
7 plt.scatter(kmeans.cluster_centers[:, 0], kmeans.cluster_centers[:,1], s = 100, c = 'yellow', label = 'Centroids')
8
9 plt.legend()
```



Market Segmentation

The screenshot shows a Jupyter Notebook titled 'python101.ipynb'. The left sidebar contains a 'Table of contents' with 'Market Segmentation' highlighted. The main area displays a Python script that imports pandas, numpy, plotly, and sklearn, then reads a CSV file from a GitHub repository. The output shows the shape of the data frame (22, 3) and a summary of its statistics, including mean, std, min, and max values for Age and Income (\$). A preview of the first five rows of the data is also shown.

```
1 # Source: https://www.kaggle.com/amanjarvis1704/k-means-clustering
2 import pandas as pd
3 import numpy as np
4 import plotly.express as px
5 import matplotlib.pyplot as plt
6 from sklearn.cluster import KMeans
7 %matplotlib inline
8 url='https://raw.githubusercontent.com/imamanmehrotra/Datasets/main/income_kmeans.csv'
9 df=pd.read_csv(url)
10 print(df.shape)
11 print(df.describe())
12 print(df)
13 px.scatter(data_frame=df, x='Age', y='Income($)', hover_data={'Name'})
```

(22, 3)

	Age	Income(\$)
count	22.000000	22.000000
mean	34.818182	90431.818182
std	5.901060	43505.964412
min	26.000000	45000.000000
25%	29.000000	58500.000000
50%	36.500000	67500.000000
75%	39.750000	135250.000000
max	43.000000	162000.000000

	Name	Age	Income(\$)
0	Rob	27	70000
1	Michael	29	90000
2	Mohan	29	61000
3	Ismail	28	60000
4	Kory	42	150000
5	Gautam	39	155000

Mall Customer Segmentation

The screenshot shows a Jupyter Notebook interface. On the left is a 'Table of contents' sidebar with a search icon and a list of topics. The main area displays a code cell with the following Python code:

```
1 #Source: Agustin Pugliese (2020), Clustering Model Comparison with Plotly, https://www.kaggle.com/agustinpugliese/c
2 import os
3 import numpy as np
4 import pandas as pd
5 import matplotlib.pyplot as plt
6 import seaborn as sns
7 from sklearn.cluster import KMeans, AgglomerativeClustering, AffinityPropagation, DBSCAN
8 import scipy.cluster.hierarchy as sch
9 import plotly.figure_factory as ff
10 import plotly.express as px
11 import plotly.graph_objects as go
12 sns.set()
13 %matplotlib inline
14 url='https://web.ntpu.edu.tw/~myday/data/example/Mall_Customers.csv'
15 df=pd.read_csv(url)
16 print(df.shape)
17 print(df.describe())
18 print(df)
19 px.scatter(data_frame=df, x='Age',y='Annual Income (k$)')
```

Below the code, the output shows the shape of the DataFrame and a summary statistics table:

```
(200, 5)
```

	CustomerID	Age	Annual Income (k\$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000

Mall Customer Segmentation



Mall Customer Segmentation

python101.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

Comment Share Settings

RAM Disk Editing

Table of contents

- Python Pandas
- Python Data Visualization
- Unsupervised Learning: Association Analysis, Market Basket Analysis
 - Association Rules Generation from Frequent Itemsets
 - Market Basket Analysis
- Unsupervised Learning: Cluster Analysis, Market Segmentation
 - Cluster Analysis: K-Means Clustering
 - Market Segmentation
 - Mall Customer Segmentation**
- Machine Learning with scikit-learn
 - Classification and Prediction
 - K-Means Clustering
- Deep Learning for Financial Time Series Forecasting
- Portfolio Optimization and Algorithmic Trading
 - Investment Portfolio Optimisation with Python
 - Efficient Frontier Portfolio Optimisation in Python
 - Investment Portfolio Optimization
- Text Analytics and Natural Language Processing (NLP)

```
1 kmeans = KMeans(n_clusters = 5, init="k-means++", max_iter = 500, n_init = 10, random_state = 123)
2 identified_clusters = kmeans.fit_predict(X)
3
4 data_with_clusters = df2.copy()
5 data_with_clusters['Cluster'] = identified_clusters
6 fig = px.scatter_3d(data_with_clusters, x = 'Age', y='Annual Income (k$)', z='Spending Score (1-100)',
7                    color='Cluster', opacity = 0.8, size='Age', size_max=30)
8 fig.show()
```

Spending Score (1-100)

Annual Income (k\$)

Age

Cluster

4

3.5

3

2.5

2

1.5

1

0.5

0

**Machine Learning:
Unsupervised Learning:
Association Analysis,
Market Basket Analysis**

Machine Learning: Data Mining Tasks & Methods

**Unsupervised Learning:
Association Analysis
Market-basket**

Association

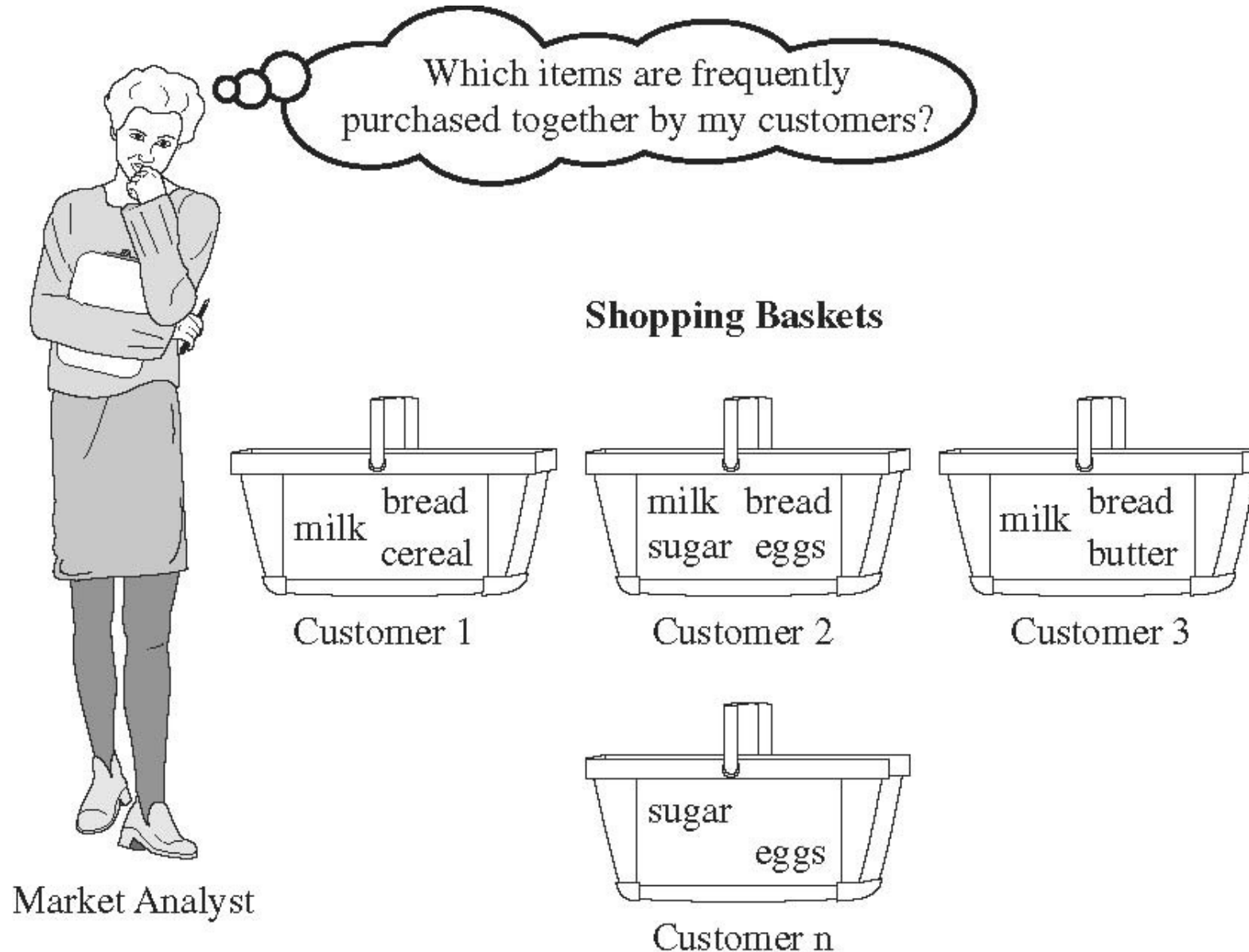
Data Mining Tasks & Methods	Data Mining Algorithms	Learning Type
Prediction		
Classification	Decision Trees, Neural Networks, Support Vector Machines, kNN, Naïve Bayes, GA	Supervised
Regression	Linear/Nonlinear Regression, ANN, Regression Trees, SVM, kNN, GA	Supervised
Time series	Autoregressive Methods, Averaging Methods, Exponential Smoothing, ARIMA	Supervised
Association		
Market-basket	Apriori, OneR, ZeroR, Eclat, GA	Unsupervised
Link analysis	Expectation Maximization, Apriori Algorithm, Graph-Based Matching	Unsupervised
Sequence analysis	Apriori Algorithm, FP-Growth, Graph-Based Matching	Unsupervised
Segmentation		
Clustering	k-means, Expectation Maximization (EM)	Unsupervised
Outlier analysis	k-means, Expectation Maximization (EM)	Unsupervised

Transaction Database

Transaction ID	Items bought
T01	A, B, D
T02	A, C, D
T03	B, C, D, E
T04	A, B, D
T05	A, B, C, E
T06	A, C
T07	B, C, D
T08	B, D
T09	A, C, E
T10	B, D

Association Analysis

Market Basket Analysis



Python mlxtend Association Rules

```
# !pip install mlxtend
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules

dataset = [['A', 'B', 'D'],
           ['A', 'C', 'D'],
           ['B', 'C', 'D', 'E'],
           ['A', 'B', 'D'],
           ['A', 'B', 'C', 'E'],
           ['A', 'C'],
           ['B', 'C', 'D'],
           ['B', 'D'],
           ['A', 'C', 'E'],
           ['B', 'D']]

te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)
frequent_itemsets = apriori(df, min_support=0.2, use_colnames=True)
association_rules(frequent_itemsets, metric="confidence", min_threshold=0.8)
```

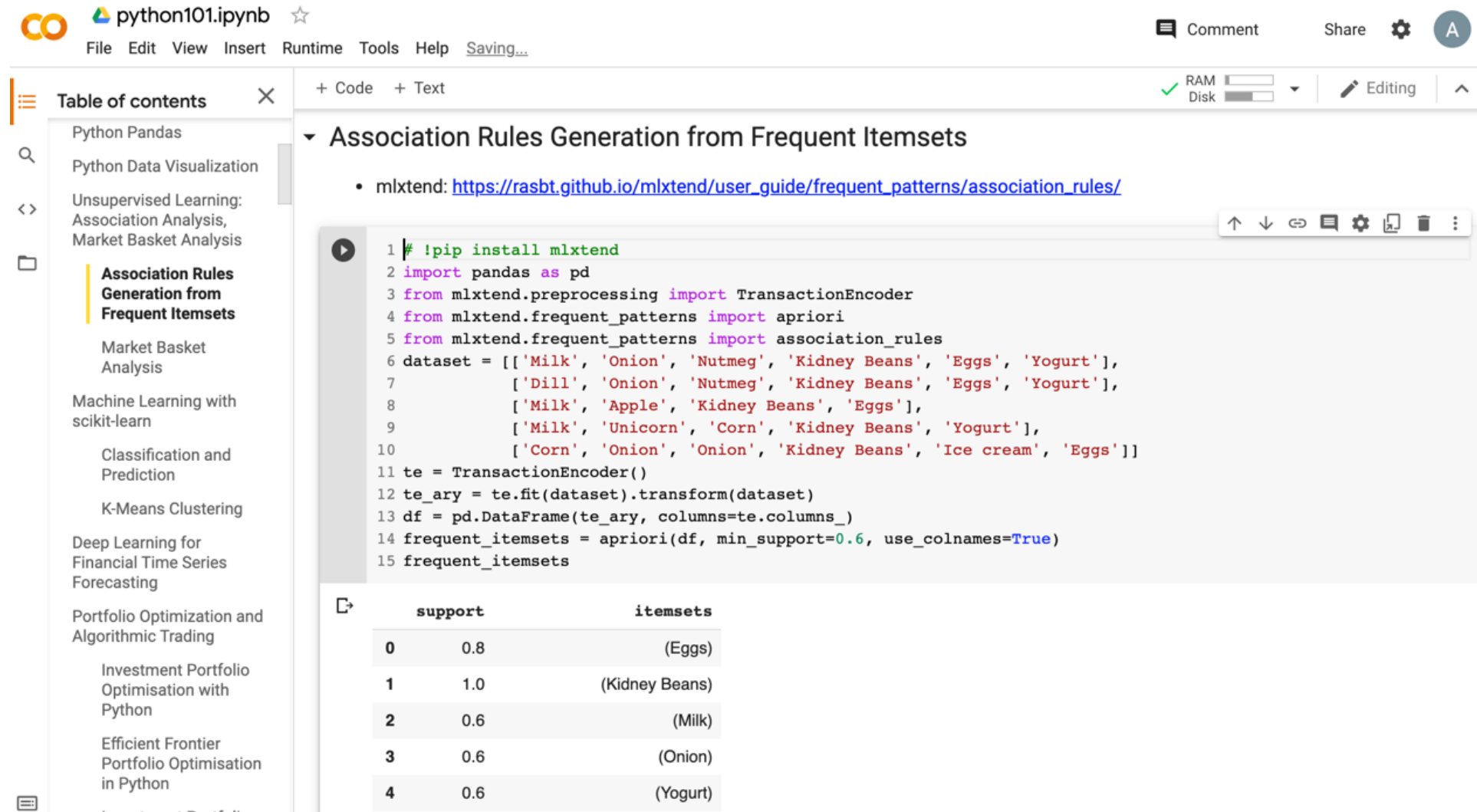
Python mlxtend Association Rules

```
1 # !pip install mlxtend
2 import pandas as pd
3 from mlxtend.preprocessing import TransactionEncoder
4 from mlxtend.frequent_patterns import apriori
5 from mlxtend.frequent_patterns import association_rules
6 dataset = [['A', 'B', 'D'],
7           ['A', 'C', 'D'],
8           ['B', 'C', 'D', 'E'],
9           ['A', 'B', 'D'],
10          ['A', 'B', 'C', 'E'],
11          ['A', 'C'],
12          ['B', 'C', 'D'],
13          ['B', 'D'],
14          ['A', 'C', 'E'],
15          ['B', 'D']]
16 te = TransactionEncoder()
17 te_ary = te.fit(dataset).transform(dataset)
18 df = pd.DataFrame(te_ary, columns=te.columns_)
19 frequent_itemsets = apriori(df, min_support=0.2, use_colnames=True)
20 rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.8)
21 rules
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(B)	(D)	0.7	0.7	0.6	0.857143	1.224490	0.11	2.1
1	(D)	(B)	0.7	0.7	0.6	0.857143	1.224490	0.11	2.1
2	(E)	(C)	0.3	0.6	0.3	1.000000	1.666667	0.12	inf
3	(E, A)	(C)	0.2	0.6	0.2	1.000000	1.666667	0.08	inf
4	(E, B)	(C)	0.2	0.6	0.2	1.000000	1.666667	0.08	inf

Python in Google Colab (Python101)

<https://colab.research.google.com/drive/1FEG6DnGvwfUbeo4zJ1zTunjMqf2RkCrT>



The screenshot shows a Google Colab notebook interface. The top bar includes the Colab logo, the notebook name 'python101.ipynb', and a star icon. Below this is a menu with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help', along with a 'Saving...' indicator. On the right side of the top bar, there are icons for 'Comment', 'Share', a settings gear, and a user profile icon 'A'. Below the top bar, there are status indicators for 'RAM' and 'Disk' usage, and a 'Editing' mode indicator.

The main content area is titled 'Association Rules Generation from Frequent Itemsets'. It contains a list item with a link to the mlxtend user guide: https://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/.

Below the link is a code cell with the following Python code:

```
1 # !pip install mlxtend
2 import pandas as pd
3 from mlxtend.preprocessing import TransactionEncoder
4 from mlxtend.frequent_patterns import apriori
5 from mlxtend.frequent_patterns import association_rules
6 dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
7            ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
8            ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
9            ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
10           ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]
11 te = TransactionEncoder()
12 te_ary = te.fit(dataset).transform(dataset)
13 df = pd.DataFrame(te_ary, columns=te.columns_)
14 frequent_itemsets = apriori(df, min_support=0.6, use_colnames=True)
15 frequent_itemsets
```

The output of the code is a table with the following data:

	support	itemsets
0	0.8	(Eggs)
1	1.0	(Kidney Beans)
2	0.6	(Milk)
3	0.6	(Onion)
4	0.6	(Yogurt)

<https://tinyurl.com/aintpupython101>

```
# ! pip install mlxtend
```

```
from mlxtend.frequent_patterns import apriori  
from mlxtend.frequent_patterns import association_rules
```

```
frequent_itemsets = apriori(df, min_support=0.6,  
use_colnames=True)
```

```
1 # ! pip install mlxtend  
2 import pandas as pd  
3 from mlxtend.preprocessing import TransactionEncoder  
4 from mlxtend.frequent_patterns import apriori  
5 from mlxtend.frequent_patterns import association_rules  
6  
7 dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],  
8           ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],  
9           ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],  
10          ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],  
11          ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]  
12  
13 te = TransactionEncoder()  
14 te_ary = te.fit(dataset).transform(dataset)  
15 df = pd.DataFrame(te_ary, columns=te.columns_)  
16 frequent_itemsets = apriori(df, min_support=0.6, use_colnames=True)  
17  
18 frequent_itemsets
```



```
# ! pip install mlxtend

import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules

dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
           ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
           ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]

te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)
frequent_itemsets = apriori(df, min_support=0.6,
                             use_colnames=True)

frequent_itemsets
```

```
frequent_itemsets = apriori(df,  
min_support=0.6,  
use_colnames=True)
```

	support	itemsets
0	0.8	(Eggs)
1	1.0	(Kidney Beans)
2	0.6	(Milk)
3	0.6	(Onion)
4	0.6	(Yogurt)
5	0.8	(Eggs, Kidney Beans)
6	0.6	(Onion, Eggs)
7	0.6	(Milk, Kidney Beans)
8	0.6	(Onion, Kidney Beans)
9	0.6	(Yogurt, Kidney Beans)
10	0.6	(Onion, Eggs, Kidney Beans)

```
association_rules(frequent_itemsets,  
metric="confidence", min_threshold=0.7)
```

```
1 association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7).
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Eggs)	(Kidney Beans)	0.8	1.0	0.8	1.00	1.00	0.00	inf
1	(Kidney Beans)	(Eggs)	1.0	0.8	0.8	0.80	1.00	0.00	1.000000
2	(Onion)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf
3	(Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000
4	(Milk)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	inf
5	(Onion)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	inf
6	(Yogurt)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	inf
7	(Onion, Eggs)	(Kidney Beans)	0.6	1.0	0.6	1.00	1.00	0.00	inf
8	(Onion, Kidney Beans)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf
9	(Eggs, Kidney Beans)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000
10	(Onion) (Eggs, Kidney Beans)		0.6	0.8	0.6	1.00	1.25	0.12	inf
11	(Eggs) (Onion, Kidney Beans)		0.8	0.6	0.6	0.75	1.25	0.12	1.600000

```
rules =  
association_rules(frequent_itemsets,  
metric="lift", min_threshold=1.2)  
rules
```

```
1 rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.2)  
2 rules
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Onion)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf
1	(Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000
2	(Onion, Kidney Beans)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf
3	(Eggs, Kidney Beans)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000
4	(Onion)	(Eggs, Kidney Beans)	0.6	0.8	0.6	1.00	1.25	0.12	inf
5	(Eggs)	(Onion, Kidney Beans)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000

```
rules["antecedent_len"] =
rules["antecedents"].apply(lambda x: len(x))
rules
```

```
1 rules["antecedent_len"] = rules["antecedents"].apply(lambda x: len(x))
2 rules
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	antecedent_len
0	(Onion)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf	1
1	(Eggs)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000	1
2	(Onion, Kidney Beans)	(Eggs)	0.6	0.8	0.6	1.00	1.25	0.12	inf	2
3	(Eggs, Kidney Beans)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000	2
4	(Onion)	(Eggs, Kidney Beans)	0.6	0.8	0.6	1.00	1.25	0.12	inf	1
5	(Eggs)	(Onion, Kidney Beans)	0.8	0.6	0.6	0.75	1.25	0.12	1.600000	1

```
rules[ (rules[ 'antecedent_len' ] >= 2) &
       (rules[ 'confidence' ] > 0.75) &
       (rules[ 'lift' ] > 1.2) ]
```

```
1 rules[ (rules[ 'antecedent_len' ] >= 2) &
2       (rules[ 'confidence' ] > 0.75) &
3       (rules[ 'lift' ] > 1.2) ]
```

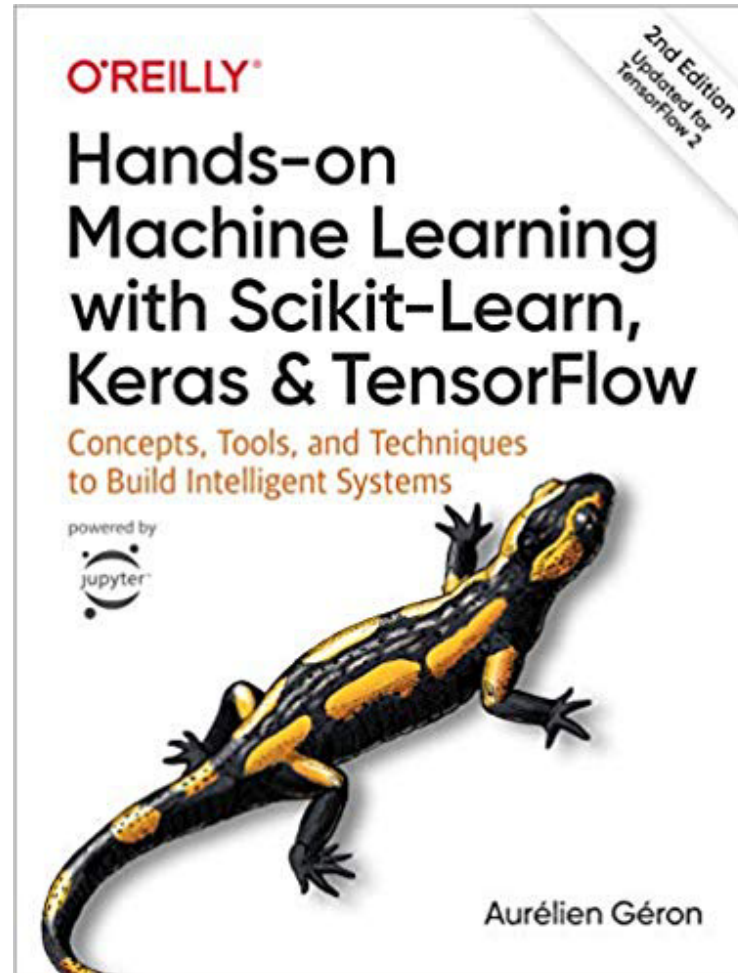
	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	antecedent_len
2	(Onion, Kidney Beans)	(Eggs)	0.6	0.8	0.6	1.0	1.25	0.12	inf	2

```
rules[rules['antecedents'] ==  
{ 'Eggs', 'Kidney Beans' }]
```

```
1 rules[rules['antecedents'] == {'Eggs', 'Kidney Beans'}]
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	antecedent_len
3	(Eggs, Kidney Beans)	(Onion)	0.8	0.6	0.6	0.75	1.25	0.12	1.6	2

Aurélien Géron (2019),
Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow:
Concepts, Tools, and Techniques to Build Intelligent Systems, 2nd Edition
O'Reilly Media, 2019



<https://github.com/ageron/handson-ml2>

Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

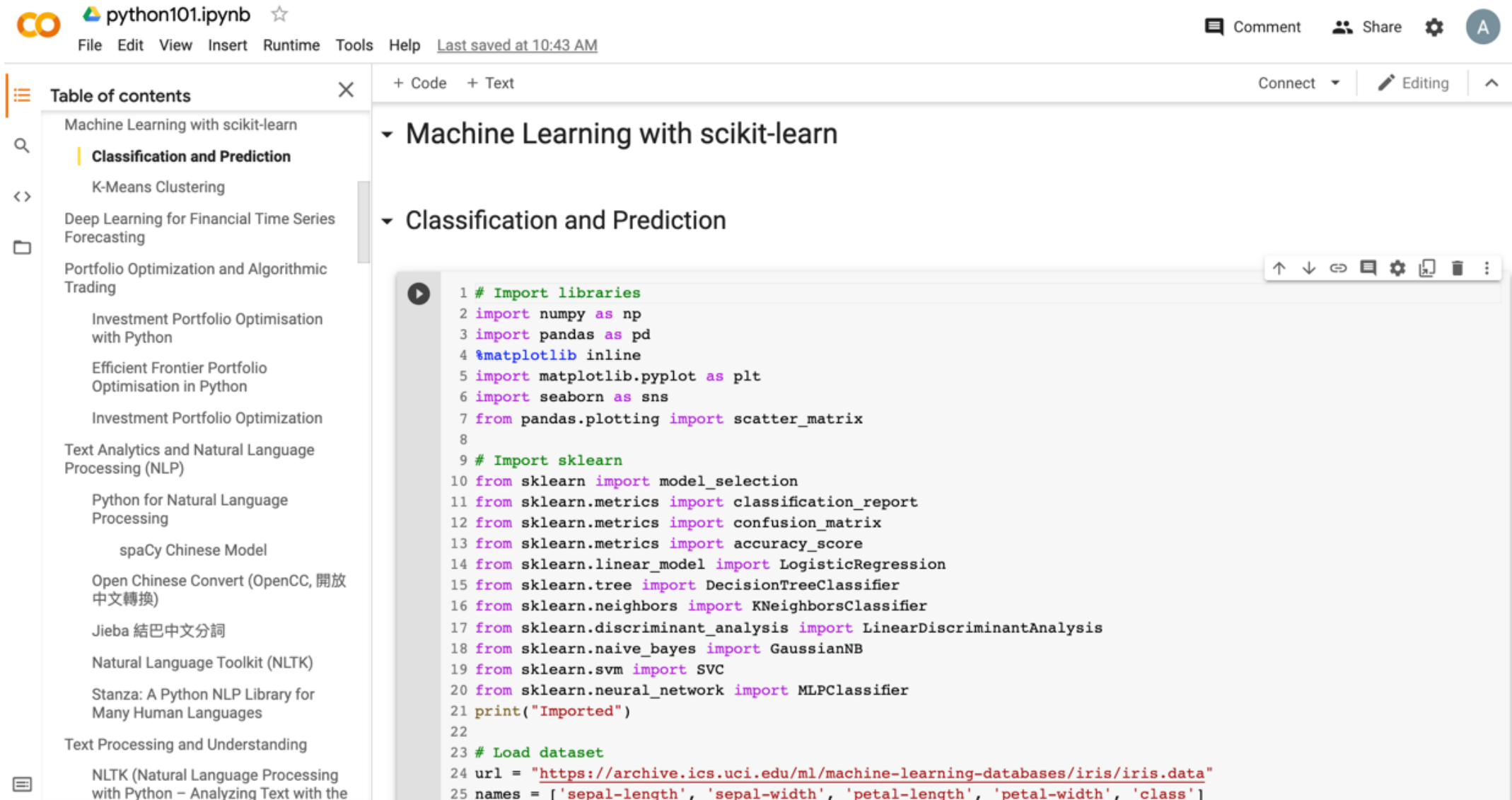
Notebooks

- [1. The Machine Learning landscape](#)
- [2. End-to-end Machine Learning project](#)
- [3. Classification](#)
- [4. Training Models](#)
- [5. Support Vector Machines](#)
- [6. Decision Trees](#)
- [7. Ensemble Learning and Random Forests](#)
- [8. Dimensionality Reduction](#)
- [9. Unsupervised Learning Techniques](#)
- [10. Artificial Neural Nets with Keras](#)
- [11. Training Deep Neural Networks](#)
- [12. Custom Models and Training with TensorFlow](#)
- [13. Loading and Preprocessing Data](#)
- [14. Deep Computer Vision Using Convolutional Neural Networks](#)
- [15. Processing Sequences Using RNNs and CNNs](#)
- [16. Natural Language Processing with RNNs and Attention](#)
- [17. Representation Learning Using Autoencoders](#)
- [18. Reinforcement Learning](#)
- [19. Training and Deploying TensorFlow Models at Scale](#)



Python in Google Colab (Python101)

<https://colab.research.google.com/drive/1FEG6DnGvwfUbeo4zJ1zTunjMqf2RkCrT>

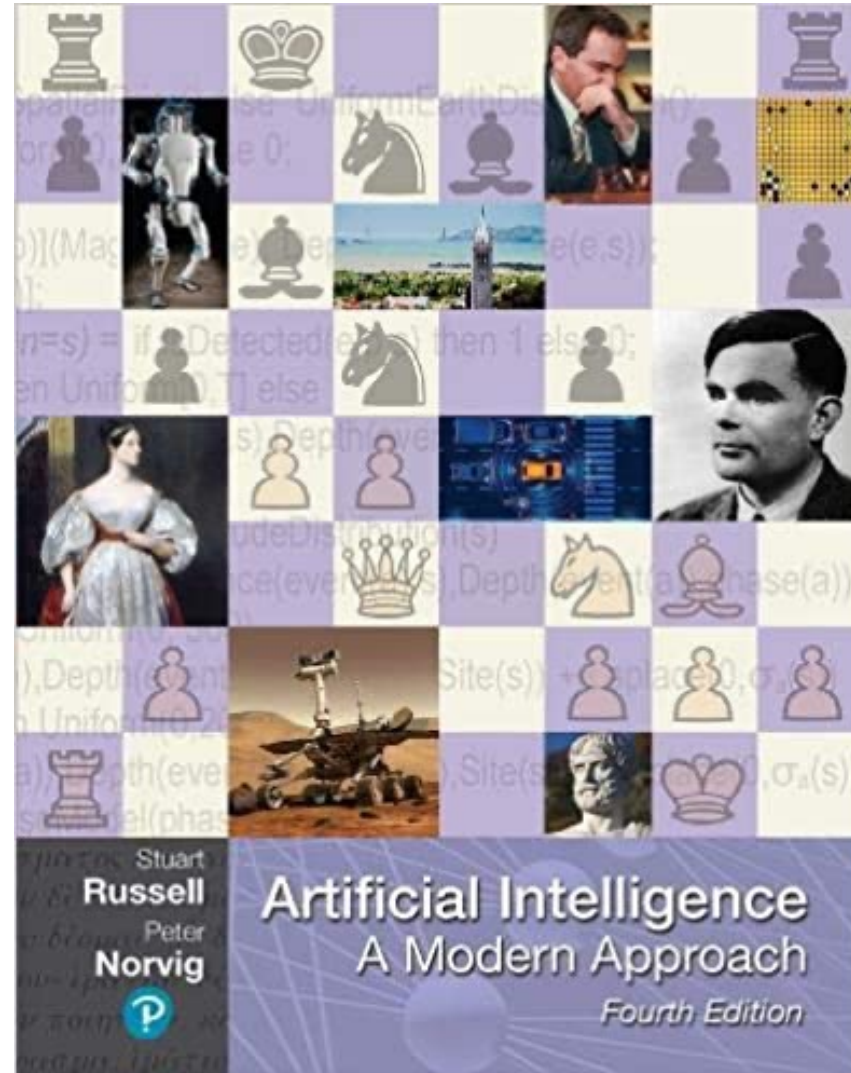


The screenshot displays the Google Colab interface for a notebook titled "python101.ipynb". The top navigation bar includes "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help", with a "Last saved at 10:43 AM" timestamp. On the right, there are icons for "Comment", "Share", "Settings", and a user profile. A "Table of contents" sidebar on the left lists various topics, with "Classification and Prediction" selected. The main workspace shows a code cell with the following Python code:

```
1 # Import libraries
2 import numpy as np
3 import pandas as pd
4 %matplotlib inline
5 import matplotlib.pyplot as plt
6 import seaborn as sns
7 from pandas.plotting import scatter_matrix
8
9 # Import sklearn
10 from sklearn import model_selection
11 from sklearn.metrics import classification_report
12 from sklearn.metrics import confusion_matrix
13 from sklearn.metrics import accuracy_score
14 from sklearn.linear_model import LogisticRegression
15 from sklearn.tree import DecisionTreeClassifier
16 from sklearn.neighbors import KNeighborsClassifier
17 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
18 from sklearn.naive_bayes import GaussianNB
19 from sklearn.svm import SVC
20 from sklearn.neural_network import MLPClassifier
21 print("Imported")
22
23 # Load dataset
24 url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
25 names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
```

<https://tinyurl.com/aintpupython101>

Stuart Russell and Peter Norvig (2020),
Artificial Intelligence: A Modern Approach,
4th Edition, Pearson



Source: Stuart Russell and Peter Norvig (2020), Artificial Intelligence: A Modern Approach, 4th Edition, Pearson

<https://www.amazon.com/Artificial-Intelligence-A-Modern-Approach/dp/0134610997/>


Artificial Intelligence: A Modern Approach (AIMA)

- **Artificial Intelligence: A Modern Approach (AIMA)**
 - <http://aima.cs.berkeley.edu/>
- **AIMA Python**
 - <http://aima.cs.berkeley.edu/python/readme.html>
 - <https://github.com/aimacode/aima-python>
- **Learning**
 - <http://aima.cs.berkeley.edu/python/learning.html>

Artificial Intelligence: A Modern Approach (AIMA)



- △ US Edition
- △ Global Edition
- Acknowledgements
- Code
- Courses
- Editions
- Errata
- Exercises
- Figures
- Instructors Page
- Pseudocode
- Reviews



Artificial Intelligence: A Modern Approach, 4th US ed.

by Stuart Russell and Peter Norvig

The authoritative, most-used AI textbook, adopted by over 1500 schools.

Table of Contents for the US Edition (or see the [Global Edition](#))

[Preface \(pdf\)](#); [Contents with subsections](#)

I Artificial Intelligence

- 1 Introduction ... 1
- 2 Intelligent Agents ... 36

II Problem-solving

- 3 Solving Problems by Searching ... 63
- 4 Search in Complex Environments ... 110
- 5 Adversarial Search and Games ... 146
- 6 Constraint Satisfaction Problems ... 180

III Knowledge, reasoning, and planning

- 7 Logical Agents ... 208
- 8 First-Order Logic ... 251
- 9 Inference in First-Order Logic ... 280
- 10 Knowledge Representation ... 314
- 11 Automated Planning ... 344

IV Uncertain knowledge and reasoning

- 12 Quantifying Uncertainty ... 385
- 13 Probabilistic Reasoning ... 412
- 14 Probabilistic Reasoning over Time ... 461
- 15 Probabilistic Programming ... 500
- 16 Making Simple Decisions ... 528
- 17 Making Complex Decisions ... 562
- 18 Multiagent Decision Making ... 599

V Machine Learning

- 19 Learning from Examples ... 651
- 20 Learning Probabilistic Models ... 721
- 21 Deep Learning ... 750
- 22 Reinforcement Learning ... 789

VI Communicating, perceiving, and acting

- 23 Natural Language Processing ... 823
- 24 Deep Learning for Natural Language Processing ... 856
- 25 Computer Vision ... 881
- 26 Robotics ... 925

VII Conclusions

- 27 Philosophy, Ethics, and Safety of AI ... 981
- 28 The Future of AI ... 1012
- Appendix A: Mathematical Background ... 1023
- Appendix B: Notes on Languages and Algorithms ... 1030
- Bibliography ... 1033 ([pdf](#) and [LaTeX .bib file](#) and [bib data](#))
- Index ... 1069 ([pdf](#))

[Exercises \(website\)](#)

[Figures \(pdf\)](#)

[Code \(website\)](#); [Pseudocode \(pdf\)](#)

Covers: [US](#), [Global](#)

Papers with Code State-of-the-Art (SOTA)



Search for papers, code and tasks



[Browse State-of-the-Art](#)

[Follow](#)

[Discuss](#)

[Trends](#)

[About](#)

[Log In/Register](#)

Browse State-of-the-Art

1509 leaderboards • 1327 tasks • 1347 datasets • 17810 papers with code

Follow on [Twitter](#) for updates

Computer Vision



Semantic Segmentation

33 leaderboards
667 papers with code



Image Classification

52 leaderboards
564 papers with code



Object Detection

54 leaderboards
467 papers with code



Image Generation

51 leaderboards
231 papers with code



Pose Estimation

40 leaderboards
231 papers with code

[See all 707 tasks](#)

Natural Language Processing



Machine Translation



Language Modelling



Question Answering



Sentiment Analysis



Text Generation

<https://paperswithcode.com/sota>

Summary

- **Machine Learning**
- **Supervised Learning**
- **Unsupervised Learning**

References

- Stuart Russell and Peter Norvig (2020), Artificial Intelligence: A Modern Approach, 4th Edition, Pearson.
- Aurélien Géron (2019), Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2nd Edition, O'Reilly Media.
- Steven D'Ascoli (2022), Artificial Intelligence and Deep Learning with Python: Every Line of Code Explained For Readers New to AI and New to Python, Independently published.
- Nithin Buduma, Nikhil Buduma, Joe Papa (2022), Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms, 2nd Edition, O'Reilly Media.