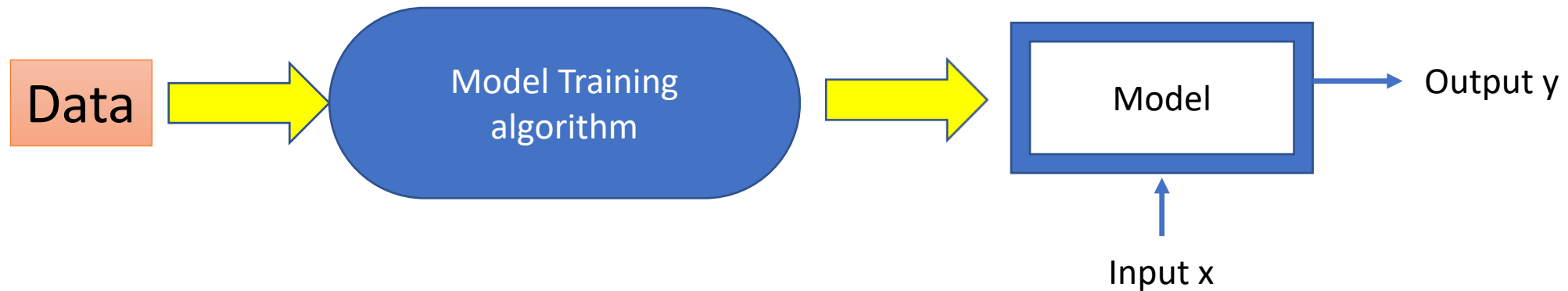


Machine Learning basics

A vertical line is positioned to the right of the text. In the bottom right corner, there is a yellow triangle pointing upwards and to the left, partially overlapping a light gray border.

Machine Learning

- It is a method that builds analytical model from data automatically.
- It is based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention.



Different Types of Machine Learning

Unsupervised Learning		Supervised Learning		Reinforcement Learning
Clustering	Consumer segmentation	Classification	Fraud detection	Real-time decisions
			Image classification	Game AI
	Recommendation System	Regression	Price prediction	Robot Navigation
			Weather forecast	Skill acquisition

Supervised Learning

- To learn a function that maps an input to an output based on a training data set that gives examples of input + label (which is the correct output) pairs
- Goal: From the training data set build a model that gives good prediction on the labels for the inputs in the dataset.
- Then, we can use this model to predict the outcome (i.e., the labels) of out-of-sample data.

Note: A model isn't necessarily a computer program, but rather, it's any mechanism that allows for a definite prediction given any input instance.

Supervised Learning: Classification

- Every input x is associated with a label y where y is from some categorical data, i.e., data whose values can be divided into (a small number) of groups.

An example

Item	Color	Shape	Label
Training data			
1		small round	Apple
2		big oval	Pear
3		small round	Apple
4		big round	Apple
:	:	:	:

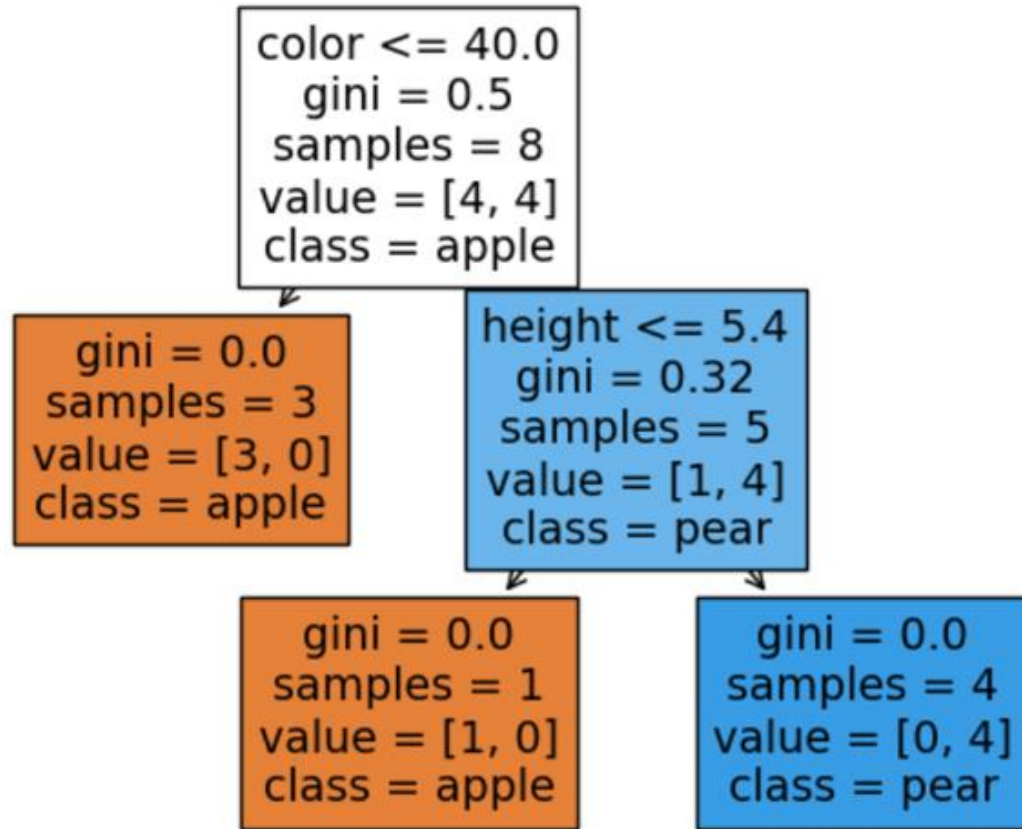
Supervised Learning: Classification

- Every input x is associated with a label y where y is from some categorical data, i.e., data whose values can be divided into (a small number) of groups.
- We need to build a model from some training data for predicting the correct label of any given input.

An example

Item	Color	Shape	Label
Training data			
1		small round	Apple
2		big oval	Pear
3		small round	Apple
4		big round	Apple
:	:	:	:
New data			
		big oval	?

Supervised Learning: Classification



An example

Item	Color	Shape	Label
Training data			
1		small round	Apple
2		big oval	Pear
3		small round	Apple
4		big round	Apple
:	:	:	:
New data			
		big oval	?

The model: Decision Tree model

Supervised learning: Regression

- Every input x is associated with a label y , where y is some numerical value (or some higher dimension numerical vector)

An example

Temperature (°C) x	Ice-cream Sales (No. of cones) y
Training data	
10	4
12	5
16	7
20	10
...	

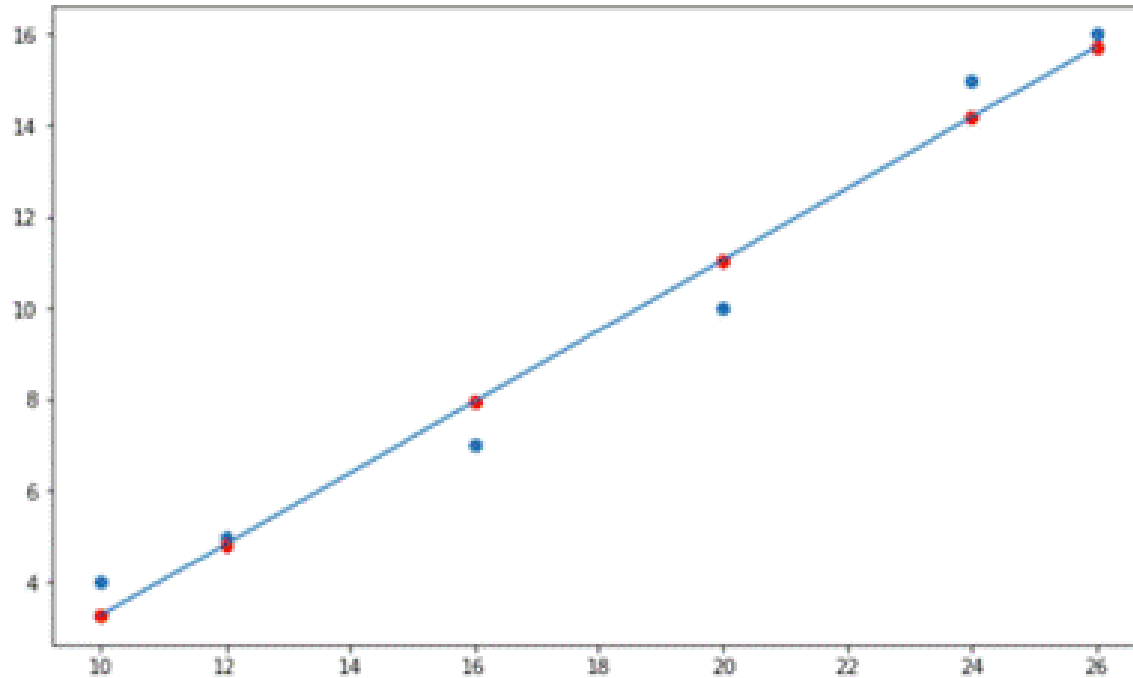
Supervised learning: Regression

- Every input x is associated with a label y , where y is some numerical value (or some higher dimension numerical vector)
- We need to build a model from some training data for predicting the correct label of any given input.

An example

Temperature (°C) x	Ice-cream Sales (No. of cones) y
Training data	
10	4
12	5
16	7
20	10
new data	
30	?

Supervised learning: Regression



The model: Linear Regression $y = b + w \cdot x$

An example

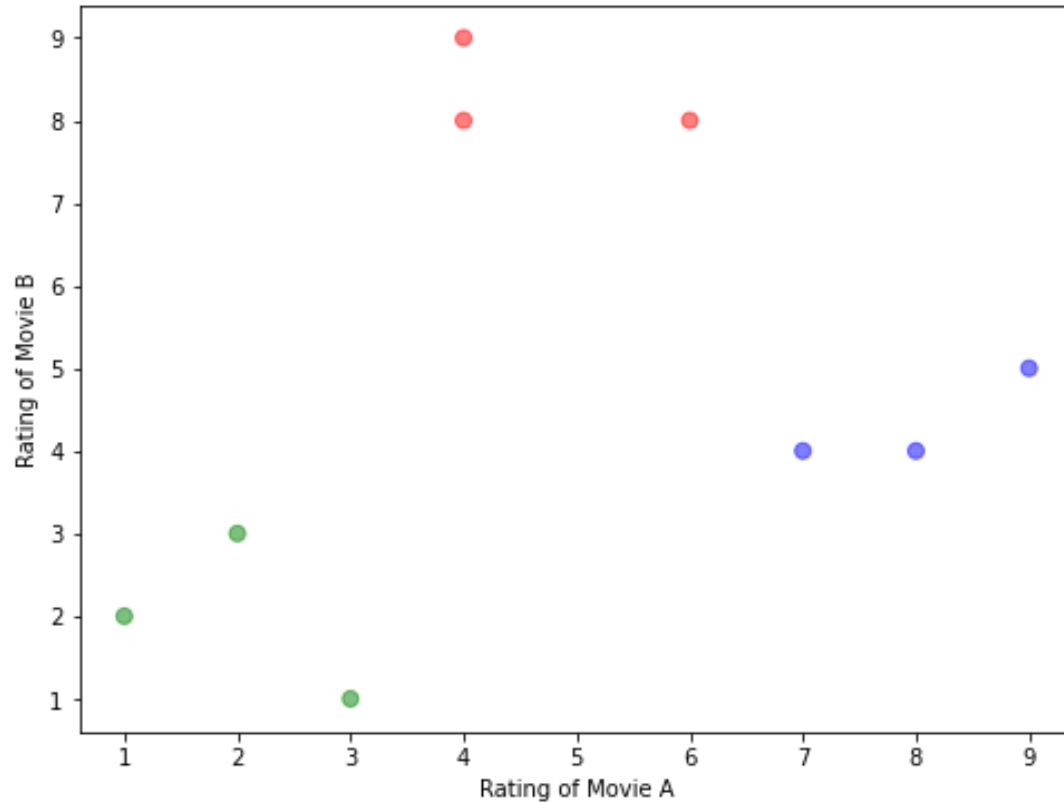
Temperature (°C) x	Ice-cream Sales (No. of cones) y
Training data	
10	4
12	5
16	7
20	10
new data	
30	?

Unsupervised Learning: Clustering

- To train and construct a model from which we can determine a **natural grouping** in data

User	Rating of Movie A	Rating of Movie B
1	1	2
2	7	4
3	2	3
4	6	8
5	4	8
6	8	4
7	9	5
8	3	1
9	4	9

Unsupervised Learning: Clustering

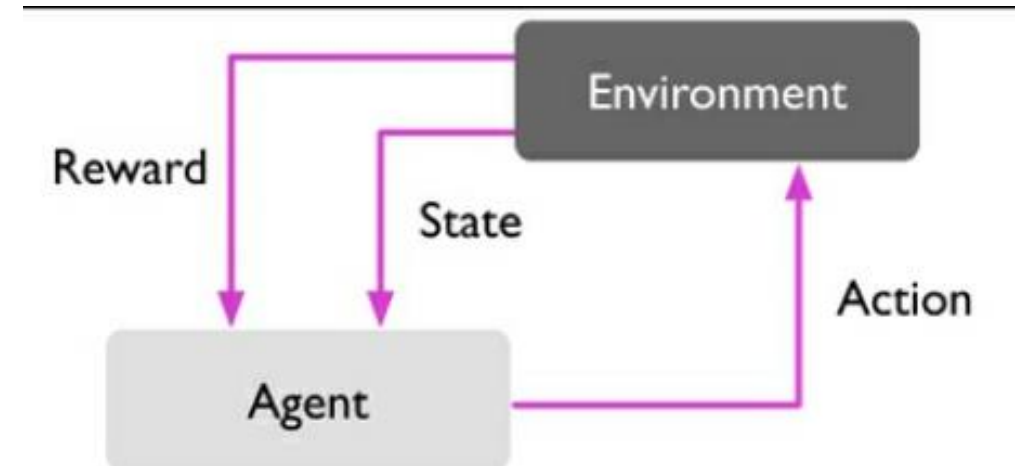


User	Rating of Movie A	Rating of Movie B
1	1	2
2	7	4
3	2	3
4	6	8
5	4	8
6	8	4
7	9	5
8	3	1
9	4	9

The model: Partition the points into groups

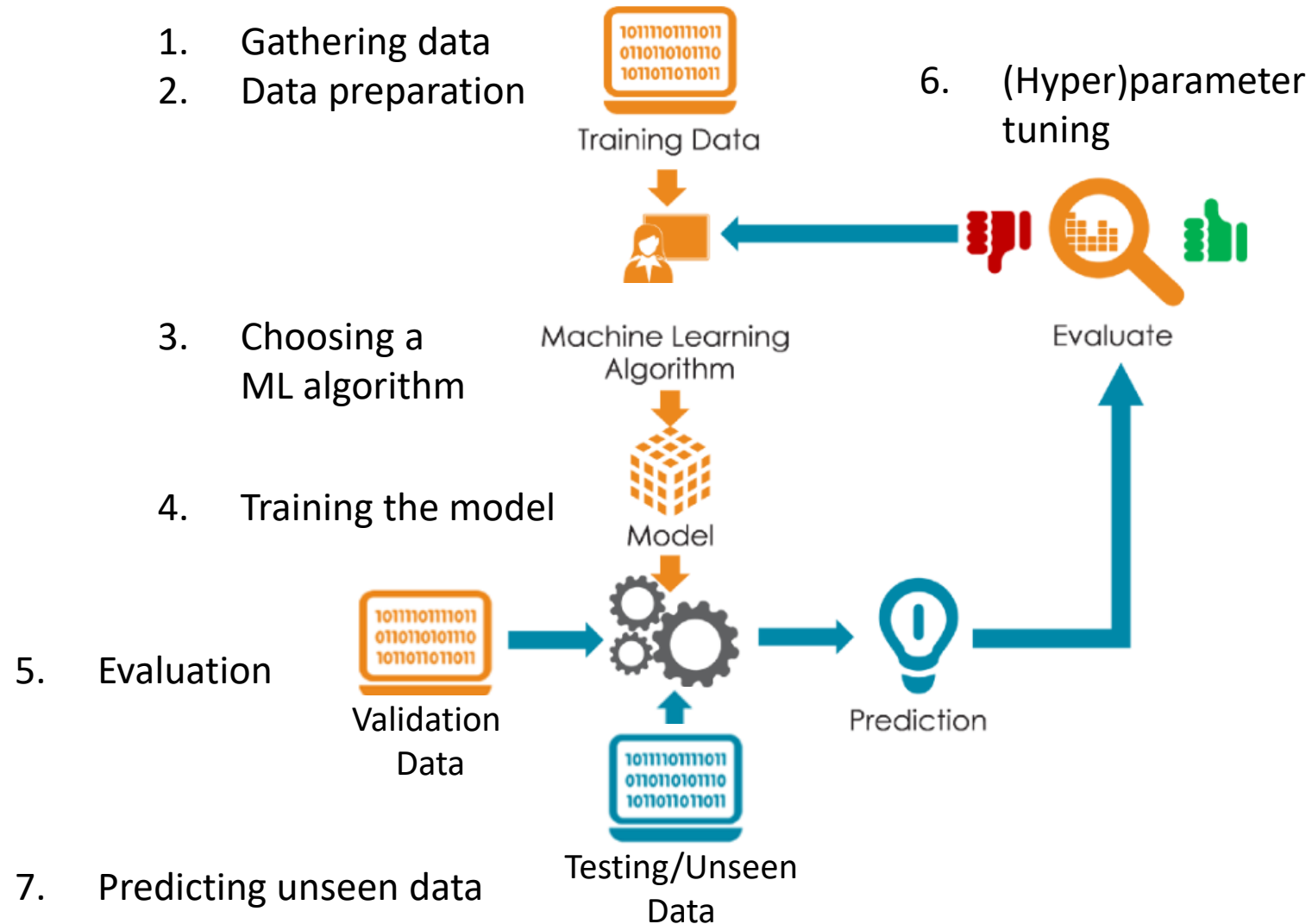
Reinforcement Learning

- Reinforcement learning is learning what to do – how to map situations to actions – so as to maximize a numerical reward signal. **The learner (agent) is not told which actions to take; instead, it must discover which actions yield the most reward by trying them.**



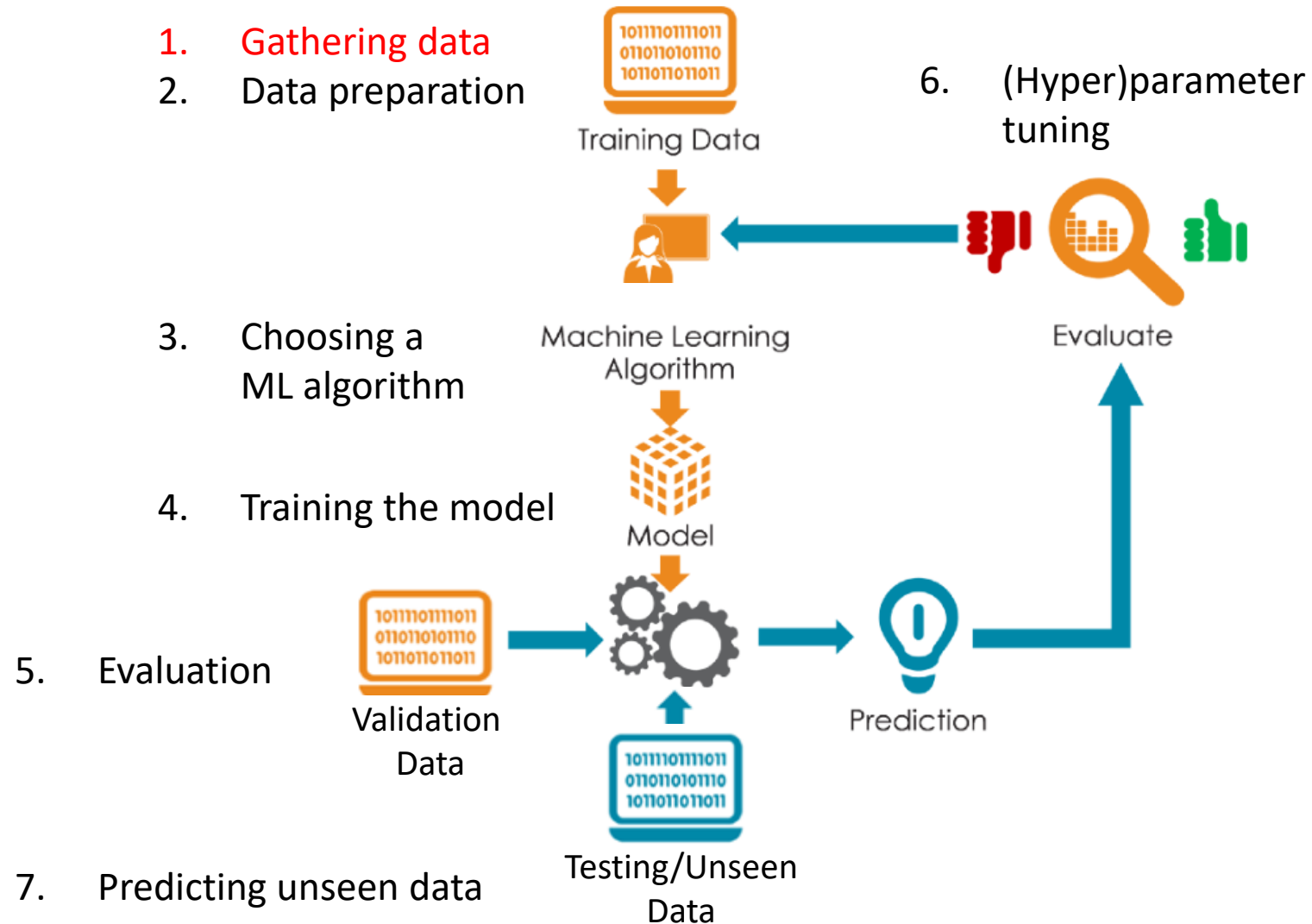
How to do Machine Learning?

7 steps of Machine Learning



How to do Machine Learning?

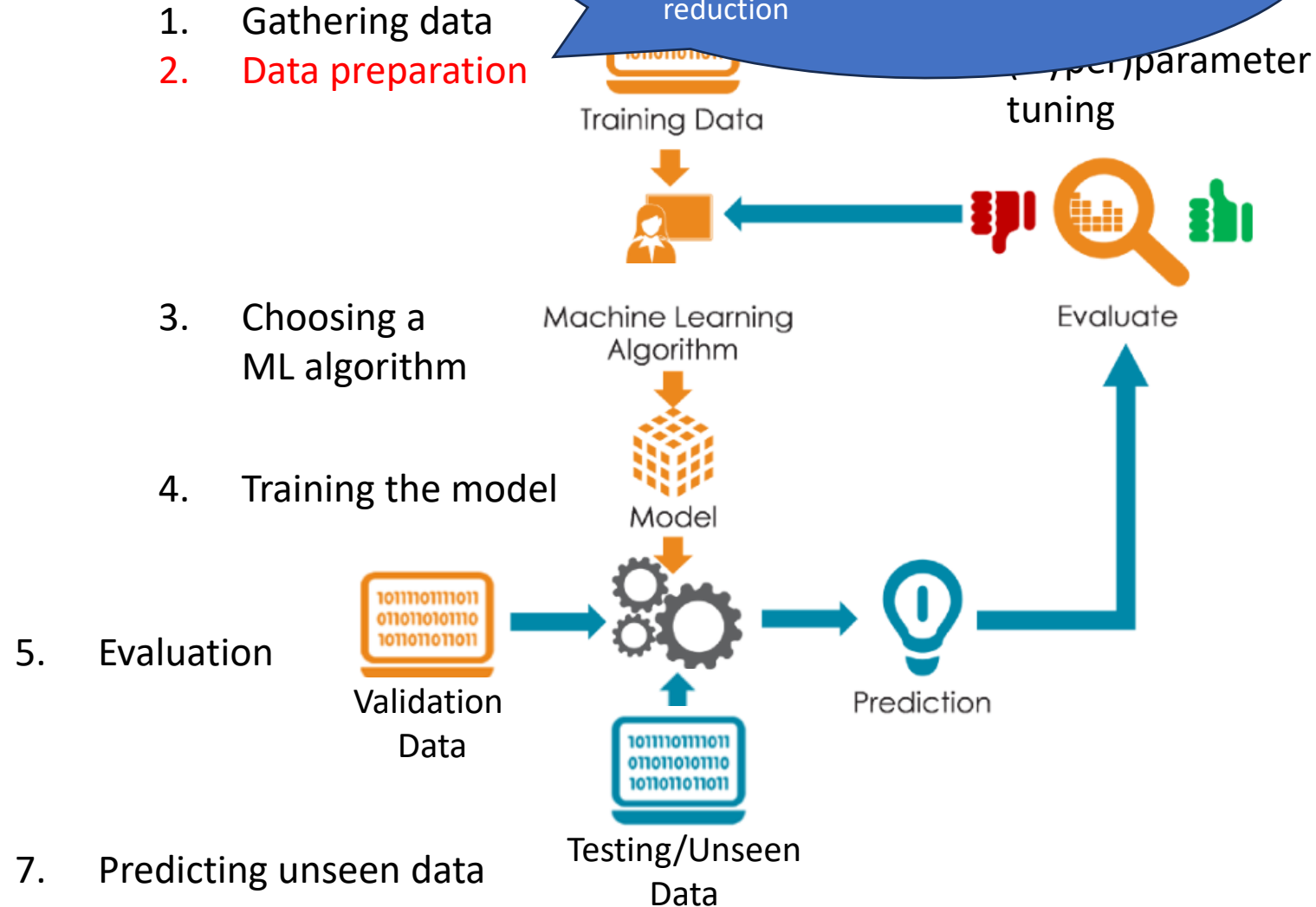
7 steps of Machine Learning



How to do Machine Learning?

7 steps of M

Handling missing values and outliers
Feature scaling
Feature selection
Feature extraction: dimensionality reduction



How to do Machine Learning?

7 steps of Machine Learning

1. Gathering data
2. Data preparation

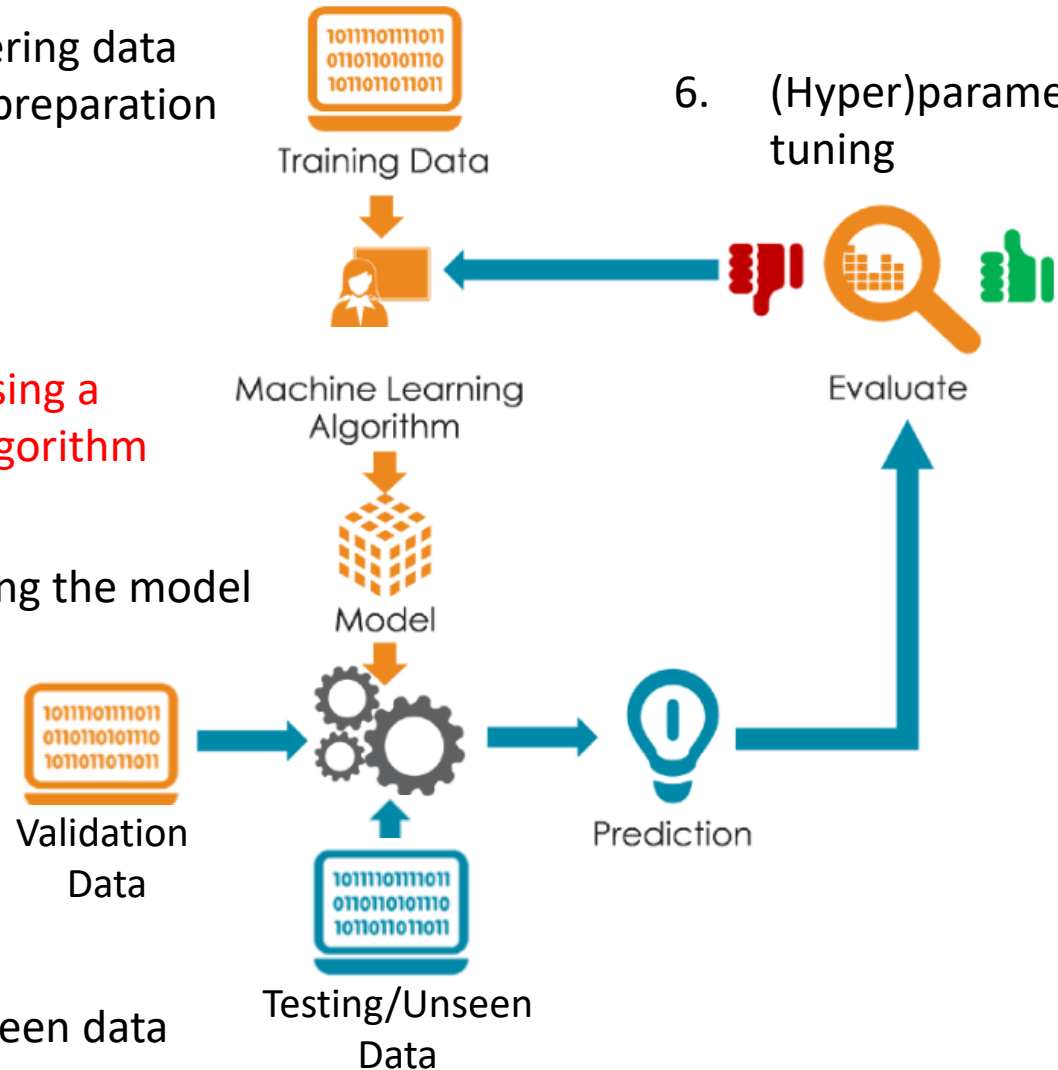
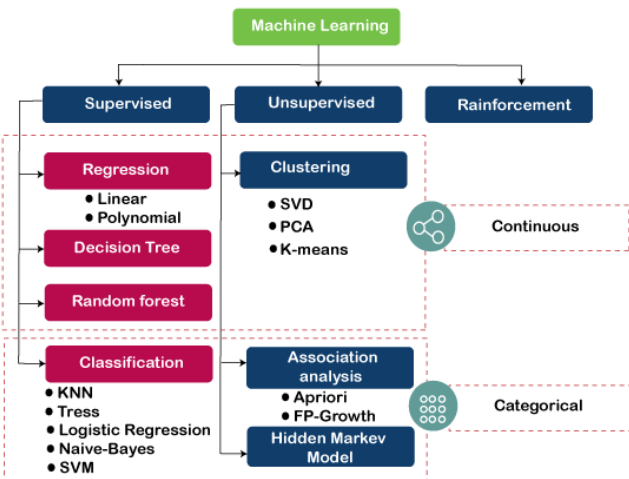
3. Choosing a ML algorithm

4. Training the model

5. Evaluation

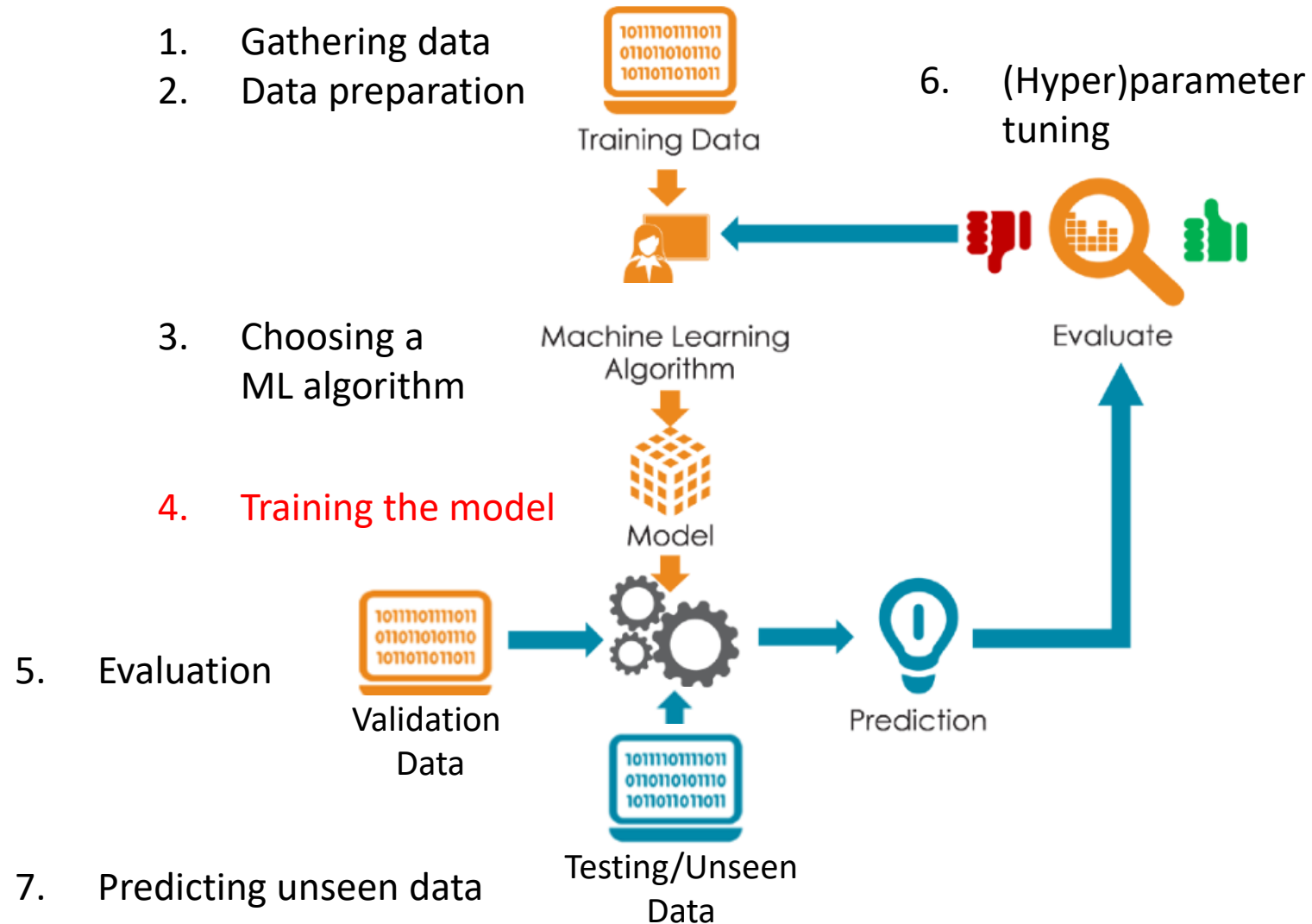
7. Predicting unseen data

6. (Hyper)parameter tuning



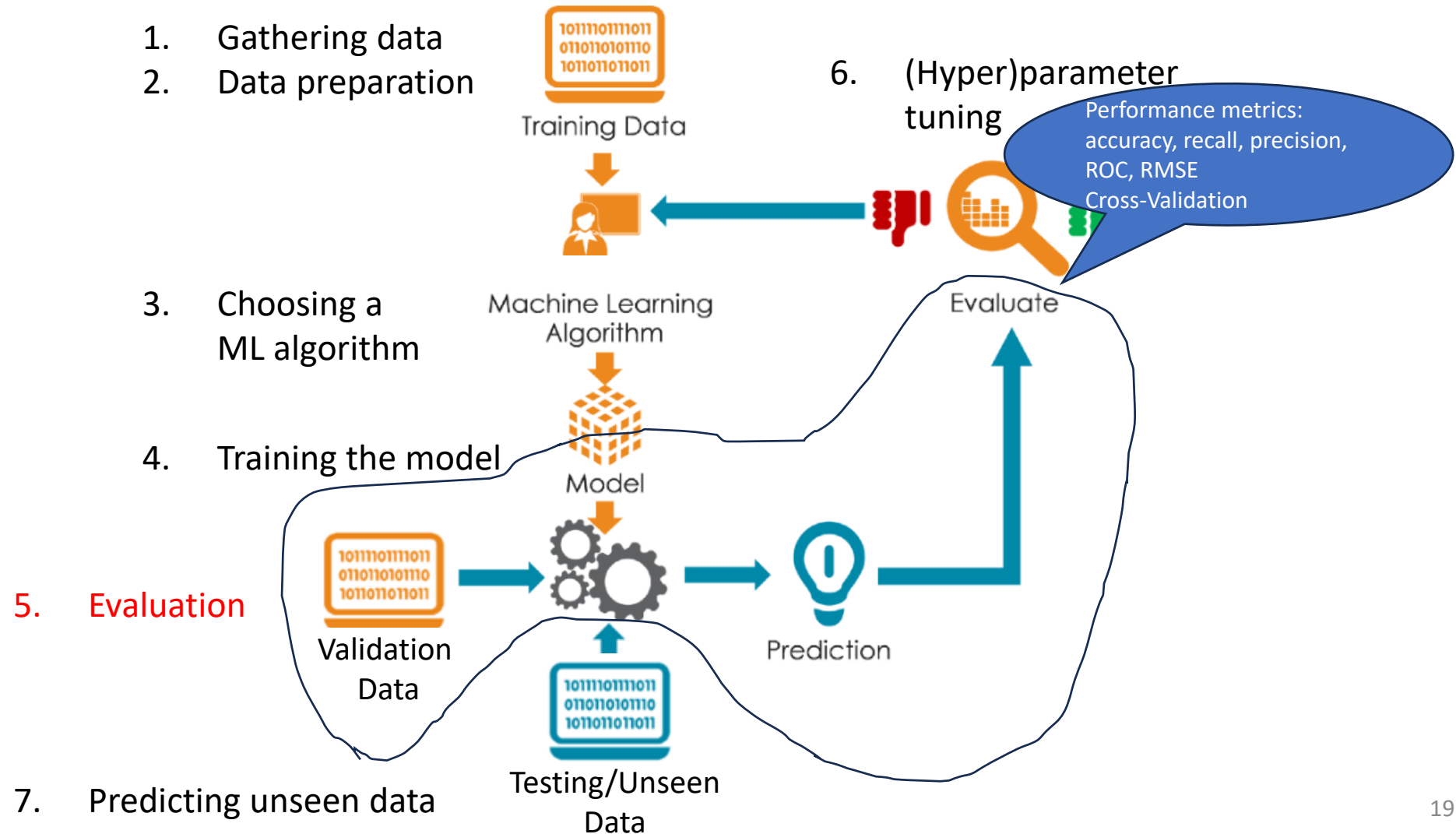
How to do Machine Learning?

7 steps of Machine Learning



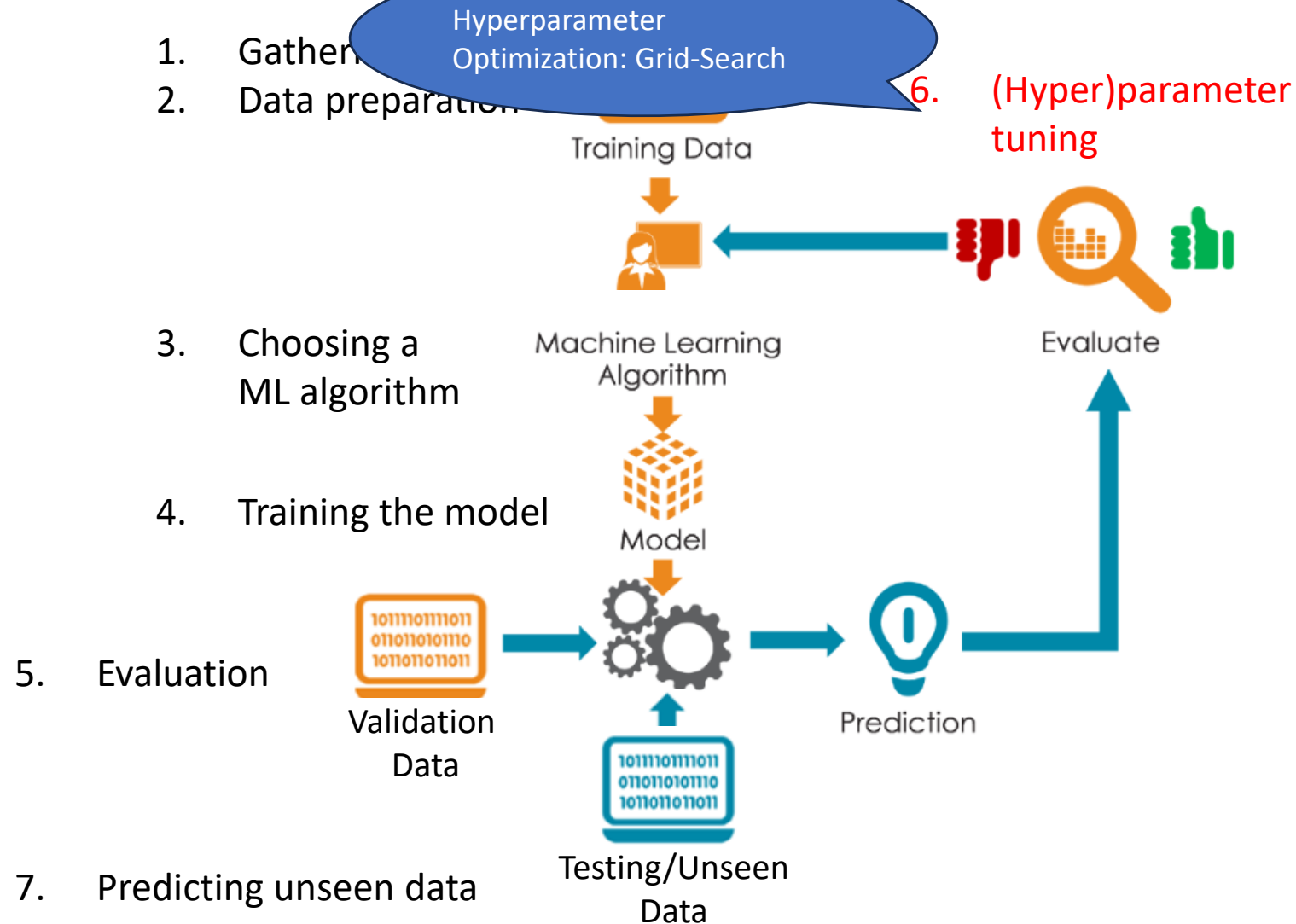
How to do Machine Learning?

7 steps of Machine Learning



How to do Machine Learning?

7 steps of Machine Learning



Performance measurement for classification

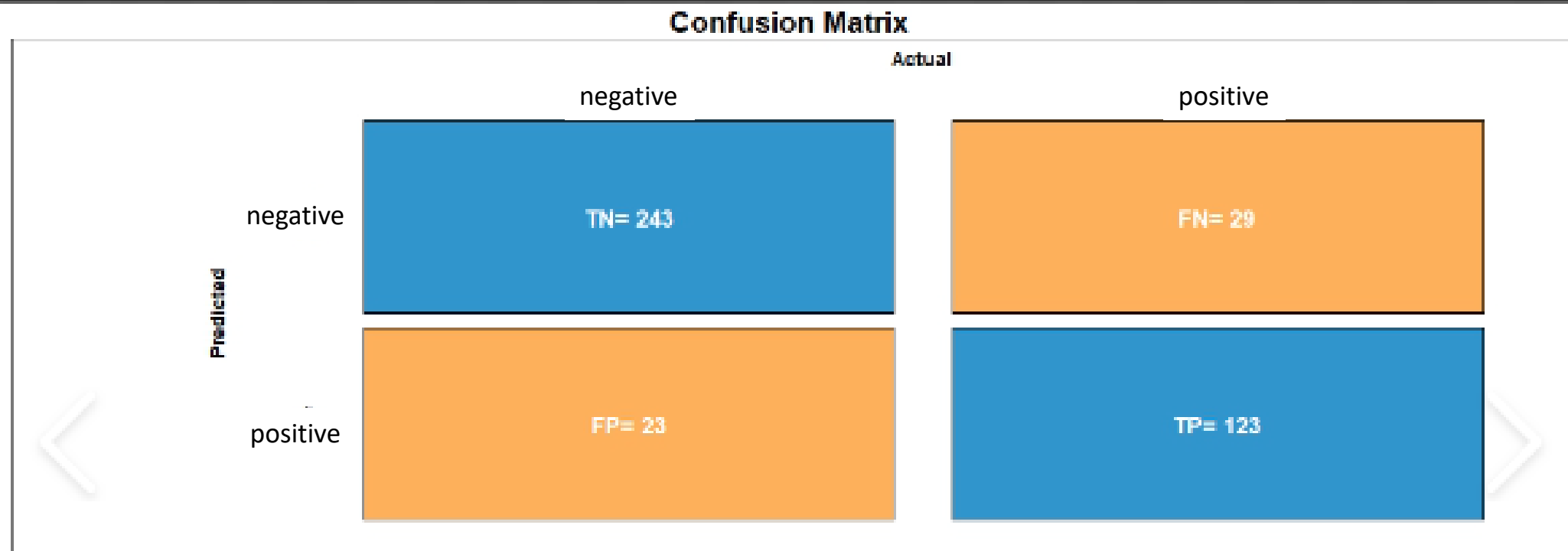
- The **accuracy** of a trained ML model for classification: The number of correct predictions (i.e., predicted outcome = correct outcome, or according to our notation, $\hat{y} = y$)
- Accuracy is a good metric for balanced classification task. But not necessarily for imbalanced ones.
 - E.g. Fraud detection in bank transactions:
 - It is no un-common that $> 99\%$ of transactions are normal, and $< 1\%$ fraud
 - If ML model does nothing but keep saying no (i.e., the transaction is not frauded), its accuracy is 99%. But it is of no use!

Better measure (for binary classification)

- Binary classification
 - Given any input, the ML algorithm predicts Positive / Negative.
E.g., Positive if the transaction is frauded; Negative it is not frauded.
- There are four possibilities
 - True Positive: actual is positive, and ML predicts a positive.
 - True Negative: actual is negative, and ML predicts a negative.
 - False Negative: actual is positive, but ML predicts a negative.
 - False Positive: actual is negative, but the test predicts a positive.

Confusion matrix

An example (which can be constructed by the `confusion_matrix(y_actual, y_pred)` method from sklearn)



Performance measures for binary classification

Name 1	Name 2	Definition
TPR	Recall	$\frac{TP}{TP+FN}$
TNR	1-FPR	$\frac{TN}{TN+FP}$
PPV	Precision	$\frac{TP}{TP+FP}$

Performance measures for binary classification

True positive rate: measures the % of positive instances that ML can detect.

Name 1	Name 2	Definition
TPR	Recall	$\frac{TP}{TP+FN}$
TNR	1-FPR	$\frac{TN}{TN+FP}$
PPV	Precision	$\frac{TP}{TP+FP}$

Performance measures for binary classification

True positive rate: measures the % of positive instances that ML can detect.

Name 1	Name 2	Formula
TPR	Recall	$\frac{TP}{TP+FN}$
TNR	1-FPR	$\frac{TN}{TN+FP}$
PPV	Precision	$\frac{TP}{TP+FP}$

True negative rate: measures the % of negative instances that ML can detect.

Positive predictive values: measures the % of correct instances that ML reports positive

Performance measures for binary classification

Other names you may find in the literature.

Name 1	Name 2	Definition
TPR	Recall	$\frac{TP}{TP+FN}$
TNR	1-FPR	$\frac{TN}{TN+FP}$
PPV	Precision	$\frac{TP}{TP+FP}$

F1-score: Another measure for binary classification

- Combine recall and precision
- Definition

$$\text{F1 Score} = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}}$$

To have a good (high) F1-score, both recall and precision have to be good.

Performance measure of regression

- Sum of Square Errors
$$SSE = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

where y_i and \hat{y}_i are the true output and the model prediction for the *ith* data point.

- Mean Square Error
$$MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

- Root Mean Square Error

$$RMSE = \sqrt{\sum_{i=1}^n \frac{(\hat{y}_i - y_i)^2}{n}}$$

Software libraries for ML

Traditional ML methods: scikit-learn



- Scikit-learn (also known as sklearn) is a free software library for Python. It includes various numerous machine learning algorithms for classification, regression and clustering.
- It was initially developed as a Google summer of code project in 2007. Later, the French institute INRIA got involved and the first public release was published in 2010.

scikit-learn

Machine Learning in Python

[Getting Started](#)[Release Highlights for 1.6](#)

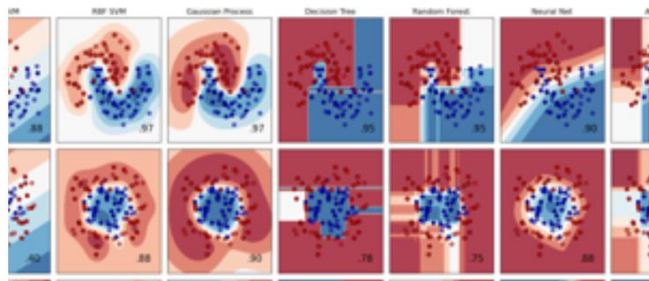
- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.

Algorithms: [Gradient boosting](#), [nearest neighbors](#), [random forest](#), [logistic regression](#), and [more...](#)

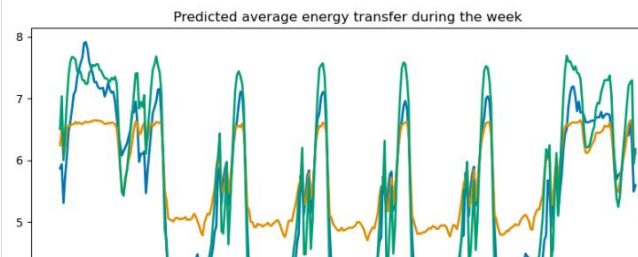


Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, stock prices.

Algorithms: [Gradient boosting](#), [nearest neighbors](#), [random forest](#), [ridge](#), and [more...](#)

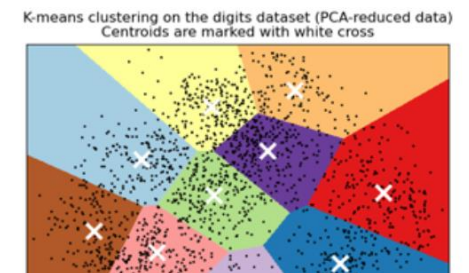


Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, grouping experiment outcomes.

Algorithms: [k-Means](#), [HDBSCAN](#), [hierarchical clustering](#), and [more...](#)





https://scikit-learn.org/stable/supervised_learning.html



Install User Guide API Examples Community More ▾

Prev Up Next

scikit-learn 1.1.1

Other versions

Please **cite us** if you use the software.

User Guide

1. Supervised learning

1.1. Linear Models
1.2. Linear and Quadratic Discriminant Analysis
1.3. Kernel ridge regression
1.4. Support Vector Machines
1.5. Stochastic Gradient Descent
1.6. Nearest Neighbors
1.7. Gaussian Processes
1.8. Cross decomposition
1.9. Naive Bayes
1.10. Decision Trees
1.11. Ensemble methods
1.12. Multiclass and multioutput algorithms
1.13. Feature selection
1.14. Semi-supervised learning
1.15. Isotonic regression
1.16. Probability calibration
1.17. Neural network models (supervised)
2. Unsupervised learning
3. Model selection and evaluation
4. Inspection
5. Visualizations
6. Dataset transformations
7. Dataset loading utilities
8. Computing with scikit-learn
9. Model persistence
10. Common pitfalls and recommended practices

1. Supervised learning

1.1. Linear Models

- 1.1.1. Ordinary Least Squares
- 1.1.2. Ridge regression and classification
- 1.1.3. Lasso
- 1.1.4. Multi-task Lasso
- 1.1.5. Elastic-Net
- 1.1.6. Multi-task Elastic-Net
- 1.1.7. Least Angle Regression
- 1.1.8. LARS Lasso
- 1.1.9. Orthogonal Matching Pursuit (OMP)
- 1.1.10. Bayesian Regression
- 1.1.11. Logistic regression
- 1.1.12. Generalized Linear Regression
- 1.1.13. Stochastic Gradient Descent - SGD
- 1.1.14. Perceptron
- 1.1.15. Passive Aggressive Algorithms
- 1.1.16. Robustness regression: outliers and modeling errors
- 1.1.17. Quantile Regression
- 1.1.18. [Polynomial regression: extending linear models with basis](#)

1.2. Linear and Quadratic Discriminant Analysis

- 1.2.1. Dimensionality reduction using Linear Discriminant Analysis
- 1.2.2. Mathematical formulation of the LDA and QDA classifiers
- 1.2.3. Mathematical formulation of LDA dimensionality reduction
- 1.2.4. Shrinkage and Covariance Estimator
- 1.2.5. Estimation algorithms

1.3. Kernel ridge regression

1.4. Support Vector Machines

- 1.4.1. Classification
- 1.4.2. Regression
- 1.4.3. Density estimation, novelty detection
- 1.4.4. Complexity
- 1.4.5. Tips on Practical Use
- 1.4.6. Kernel functions
- 1.4.7. Mathematical formulation

1.5. Stochastic Gradient Descent

- 1.5.1. Classification
- 1.5.2. Regression
- 1.5.3. Online One-Class SVM
- 1.5.4. Stochastic Gradient Descent for sparse d
- 1.5.5. Complexity
- 1.5.6. Stopping criterion
- 1.5.7. Tips on Practical Use
- 1.5.8. Mathematical formulation
- 1.5.9. Implementation details

1.6. Nearest Neighbors

- 1.6.1. Unsupervised Nearest Neighbors
- 1.6.2. Nearest Neighbors Classification
- 1.6.3. Nearest Neighbors Regression
- 1.6.4. Nearest Neighbor Algorithms
- 1.6.5. Nearest Centroid Classifier
- 1.6.6. Nearest Neighbors Transformer
- 1.6.7. Neighborhood Components Analysis

1.7. Gaussian Processes

- 1.7.1. Gaussian Process Regression (GPR)
- 1.7.2. GPR examples
- 1.7.3. Gaussian Process Classification (GPC)
- 1.7.4. GPC examples
- 1.7.5. Kernels for Gaussian Processes

1.8. Cross decomposition

- 1.8.1. PLSCanonical
- 1.8.2. PLSSVD
- 1.8.3. PLSRegression
- 1.8.4. Canonical Correlation Analysis

1.9. Naive Bayes

- 1.9.1. Gaussian Naive Bayes
- 1.9.2. Multinomial Naive Bayes
- 1.9.3. Complement Naive Bayes
- 1.9.4. Bernoulli Naive Bayes
- 1.9.5. Categorical Naive Bayes
- 1.9.6. Out-of-core naive Bayes model fitting

1.10. Decision Trees

- 1.10.1. Classification
- 1.10.2. Regression
- 1.10.3. Multi-output problems
- 1.10.4. Complexity
- 1.10.5. Tips on practical use
- 1.10.6. Tree algorithms: ID3, C4.5, C5.0 and CART
- 1.10.7. Mathematical formulation
- 1.10.8. Minimal Cost-Complexity Pruning

1.11. Ensemble methods

- 1.11.1. Bagging meta-estimator
- 1.11.2. Forests of randomized trees
- 1.11.3. AdaBoost
- 1.11.4. Gradient Tree Boosting
- 1.11.5. Histogram-Based Gradient Boosting
- 1.11.6. Voting Classifier
- 1.11.7. Voting Regressor
- 1.11.8. Stacked generalization

1.12. Multiclass and multioutput algorithms

- 1.12.1. Multiclass classification
- 1.12.2. Multilabel classification
- 1.12.3. Multiclass-multioutput classification
- 1.12.4. Multioutput regression

1.13. Feature selection

- 1.13.1. Removing features with low variance
- 1.13.2. Univariate feature selection
- 1.13.3. Recursive feature elimination
- 1.13.4. Feature selection using SelectFromModel
- 1.13.5. Sequential Feature Selection
- 1.13.6. Feature selection as part of a pipeline

1.14. Semi-supervised learning

- 1.14.1. Self Training
- 1.14.2. Label Propagation

1.15. Isotonic regression

1.16. Probability calibration

- 1.16.1. Calibration curves
- 1.16.2. Calibrating a classifier
- 1.16.3. Usage

1.17. Neural network models (supervised)

- 1.17.1. Multi-layer Perceptron
- 1.17.2. Classification
- 1.17.3. Regression
- 1.17.4. Regularization
- 1.17.5. Algorithms
- 1.17.6. Complexity
- 1.17.7. Mathematical formulation
- 1.17.8. Tips on Practical Use
- 1.17.9. More control with warm_start

1. Supervised learning

1.1. Linear Models

[1.1.1. Ordinary Least Squares](#)

[1.1.2. Ridge regression and classification](#)

[1.1.3. Lasso](#)

[1.1.4. Multi-task Lasso](#)

[1.1.5. Elastic-Net](#)

[1.1.6. Multi-task Elastic-Net](#)

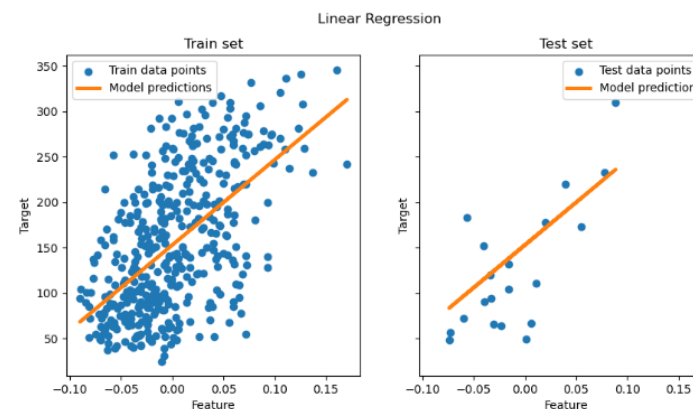
[1.1.7. Least Angle Regression](#)

[1.1.8. LARS Lasso](#)

1.1.1. Ordinary Least Squares

`LinearRegression` fits a linear model with coefficients $w = (w_1, \dots, w_p)$ to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation. Mathematically it solves a problem of the form:

$$\min_w ||Xw - y||_2^2$$



`LinearRegression` will take in its `fit` method arrays `X`, `y` and will store the coefficients w of the linear model in its `coef_` member:

```
>>> from sklearn import linear_model
>>> reg = linear_model.LinearRegression()
>>> reg.fit([[0, 0], [1, 1], [2, 2]], [0, 1, 2])
LinearRegression()
>>> reg.coef_
array([0.5, 0.5])
```

Some examples

Supervised learning: Linear Regression model for predicting ice cream sales

```
[1] import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression
```

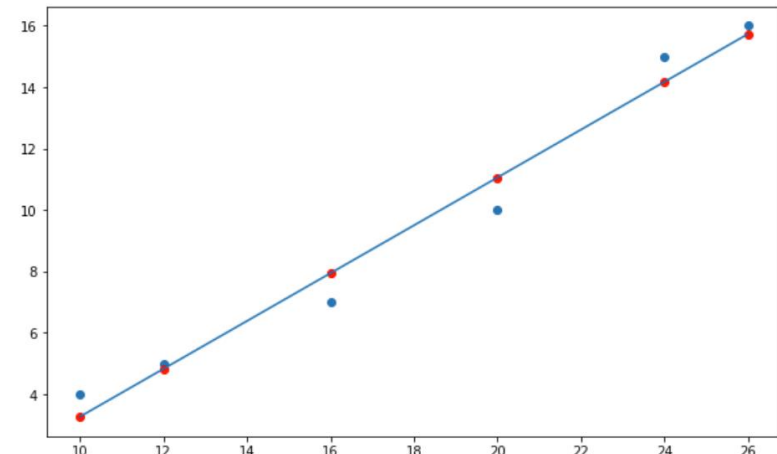
```
▶ x = np.array([10,12,16,20,24,26]).reshape((-1,1))
y = np.array([4,5,7,10,15, 16])
```

```
[3] model = LinearRegression()
model.fit(x, y)
```

```
[4] y_pred = model.predict(x)

plt.figure(figsize=(10,6))
plt.scatter(x,y)
plt.scatter(x,y_pred,color='r')
x_seq = np.linspace(x.min(),x.max(),15).reshape((-1,1))
plt.plot(x_seq, model.predict(x_seq))
plt.show()
```

Temperature (°C) x	Ice-cream Sales (No. of cones) y
10	4
12	5
16	7
20	10
24	15
30	?



Supervised learning: Linear Regression model for predicting ice cream sales

```
[1] import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression
```

Why reshape() here?

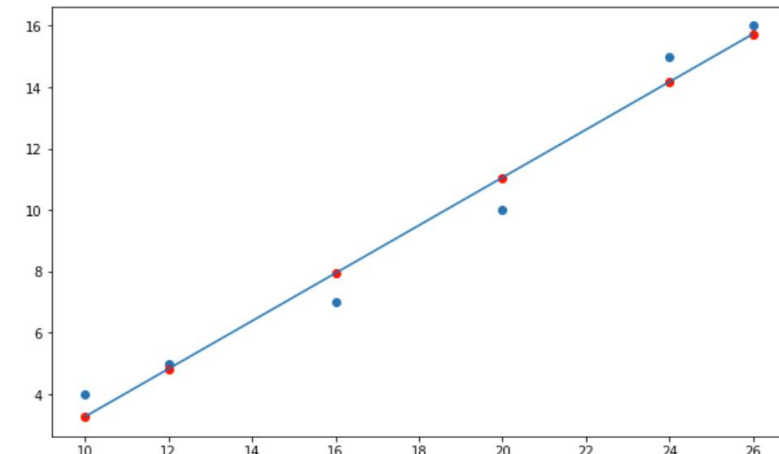
```
▶ x = np.array([10,12,16,20,24,26]).reshape((-1,1))
y = np.array([4,5,7,10,15, 16])
```

```
[3] model = LinearRegression()
model.fit(x, y)
```

```
[4] y_pred = model.predict(x)

plt.figure(figsize=(10,6))
plt.scatter(x,y)
plt.scatter(x,y_pred,color='r')
x_seq = np.linspace(x.min(),x.max(),15).reshape((-1,1))
plt.plot(x_seq, model.predict(x_seq))
plt.show()
```

Temperature (°C)	Ice-cream Sales (No. of cones)
x	y
10	4
12	5
16	7
20	10
24	15
30	?



Supervised learning: Linear Regression model for predicting ice cream sales

```
[1] import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression
```

```
▶ x = np.array([10,12,16,20,24,26]).reshape((-1,1))
y = np.array([4,5,7,10,15, 16])
```

```
[3] model = LinearRegression()
model.fit(x, y)
```

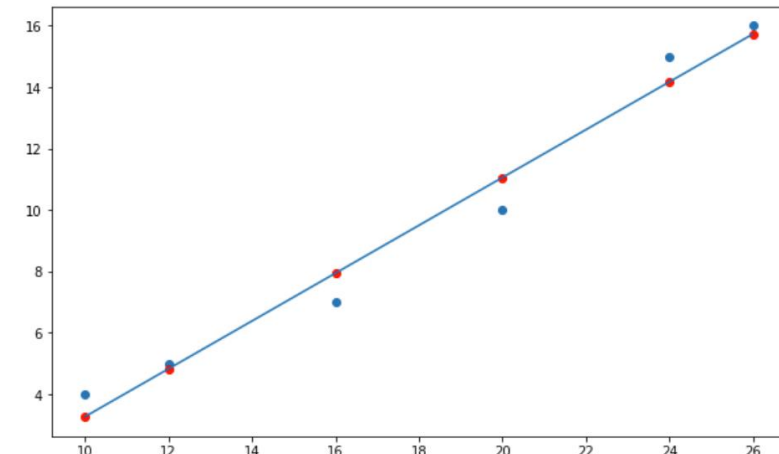
```
[4] y_pred = model.predict(x)

plt.figure(figsize=(10,6))
plt.scatter(x,y)
plt.scatter(x,y_pred,color='r')
x_seq = np.linspace(x.min(),x.max(),15).reshape((-1,1))
plt.plot(x_seq, model.predict(x_seq))
plt.show()

model.predict([[30]])

array([18.84615385])
```

Temperature (°C) x	Ice-cream Sales (No. of cones) y
10	4
12	5
16	7
20	10
24	15
30	?



Supervised learning: Classifying Apple vs Pear

```
▶ from sklearn import tree  
import numpy as np
```

```
▶ features = ['color', 'width', 'height']  
fruitnames=['apple', 'pear']  
  
data = np.array( [[10, 5.3, 4.8], [62, 5.5, 7.8], [85, 5.7, 4.9], [58, 6.8, 5.9],  
                 [15, 8.6, 7.7], [100, 5.4, 8.2], [22, 5.8, 5.6], [71, 4.5, 6.3]])  
labels = np.array([0, 1, 0, 1, 0, 1, 0, 1])
```

```
[3] fruit_classifier = tree.DecisionTreeClassifier()
```

```
[4] fruit_classifier = fruit_classifier.fit(data, labels)
```

```
[5] predict = fruit_classifier.predict(np.array([[101, 6.3, 7.9], [25, 4.5, 3.8], [85, 5.2, 4.0]]))  
predict
```

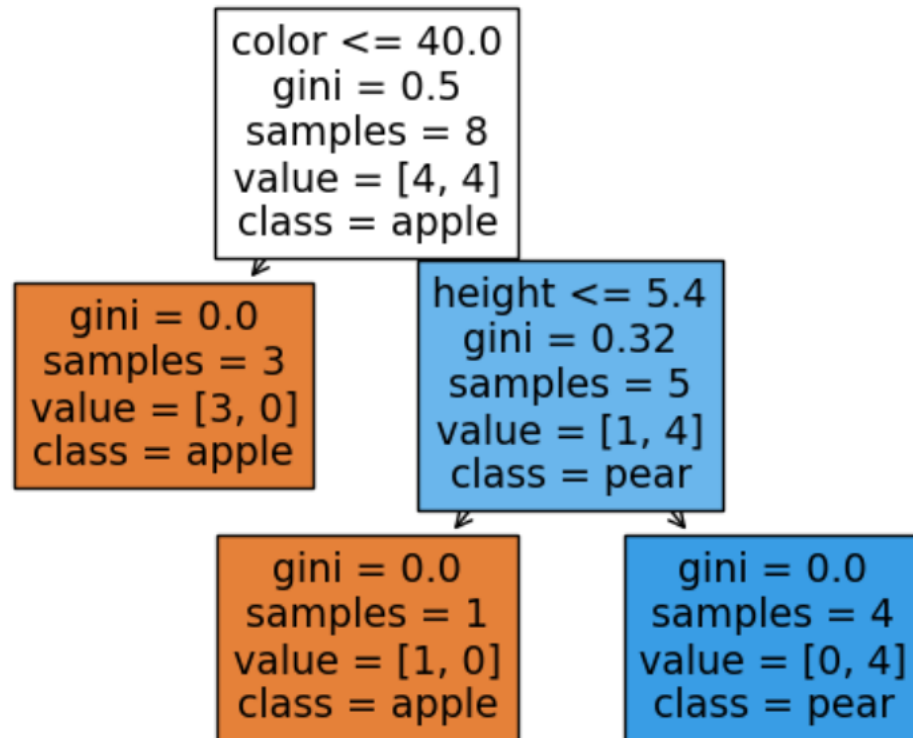
```
array([1, 0, 0])
```

```
[9] print(fruitnames[int(fruit_classifier.predict([[101, 6.3, 7.9]])[0])])  
  
pear
```

Supervised learning: Classifying Apple vs Pear

```
▶ tree.plot_tree(fruit_classifier, feature_names = features, class_names = fruit_names,  
                filled = True)
```

```
↳ [Text(0.4, 0.8333333333333334, 'color <= 40.0\ngini = 0.5\nsamples = 8\nvalue = [4, 4]\nclass = apple')  
   Text(0.2, 0.5, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]\nclass = apple'),  
   Text(0.6, 0.5, 'height <= 5.4\ngini = 0.32\nsamples = 5\nvalue = [1, 4]\nclass = pear'),  
   Text(0.4, 0.16666666666666666, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]\nclass = apple'),  
   Text(0.8, 0.16666666666666666, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]\nclass = pear')]
```



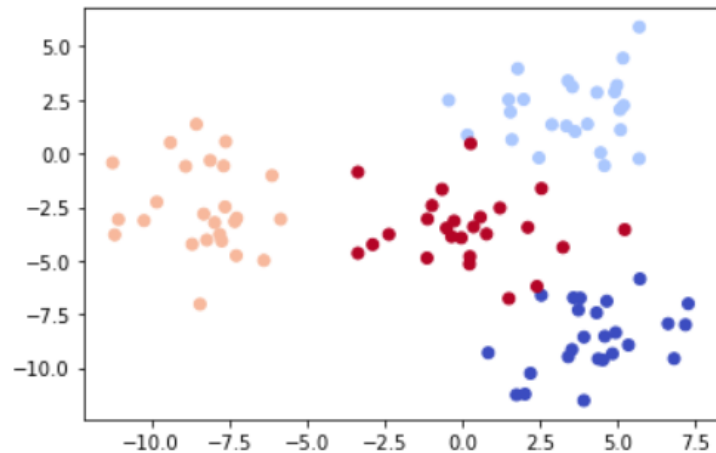
Unsupervised learning: clustering points on 2D plane

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

```
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
```

```
x, y = make_blobs(n_samples=100, centers=4, random_state=500, cluster_std=1.75)
```

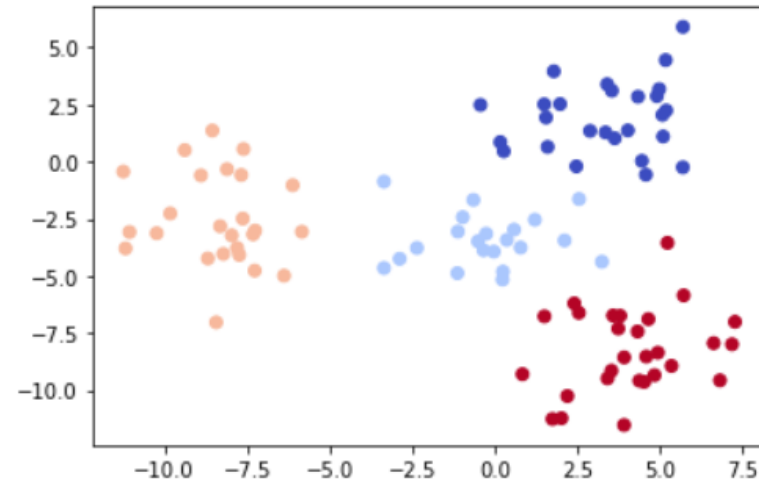
```
plt.scatter(x[:,0],x[:,1], c=y, cmap='coolwarm')
plt.show()
```



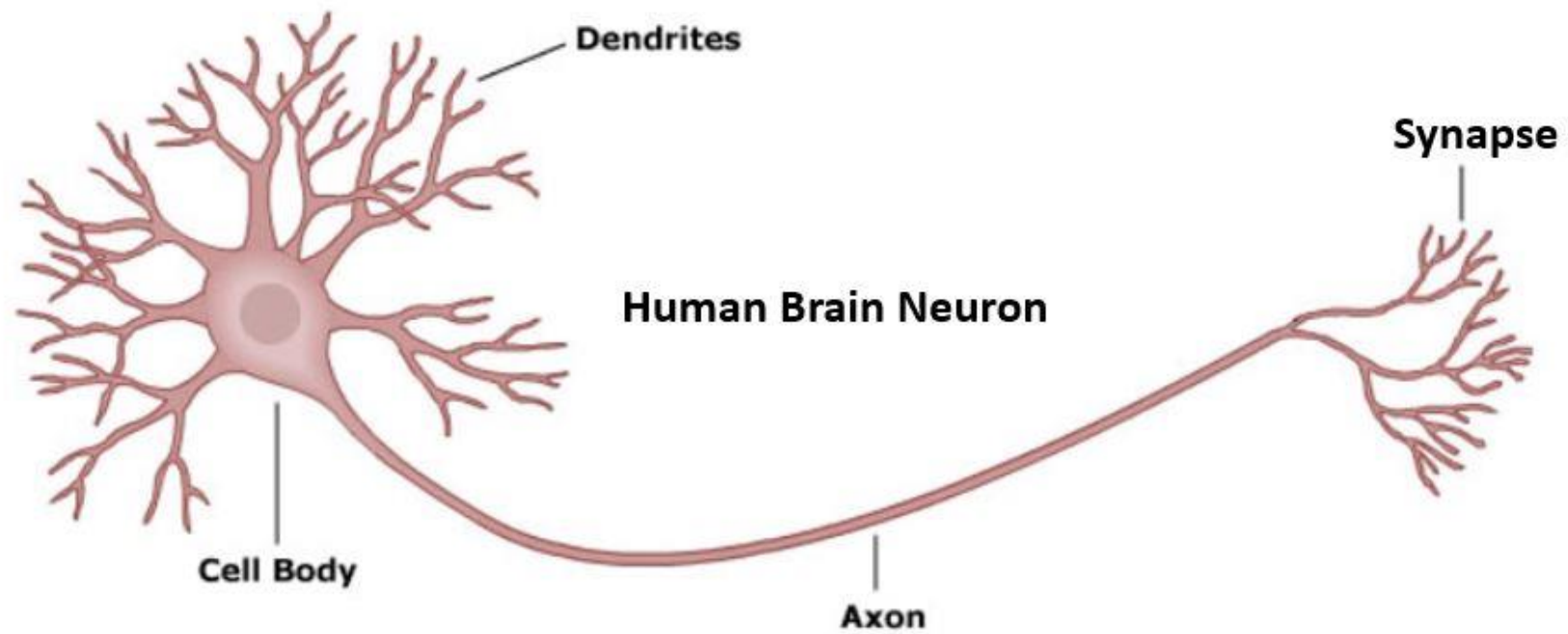
```
model = KMeans(n_clusters=4, random_state = 0)
model.fit(x)
```

```
y_ = model.predict(x)
```

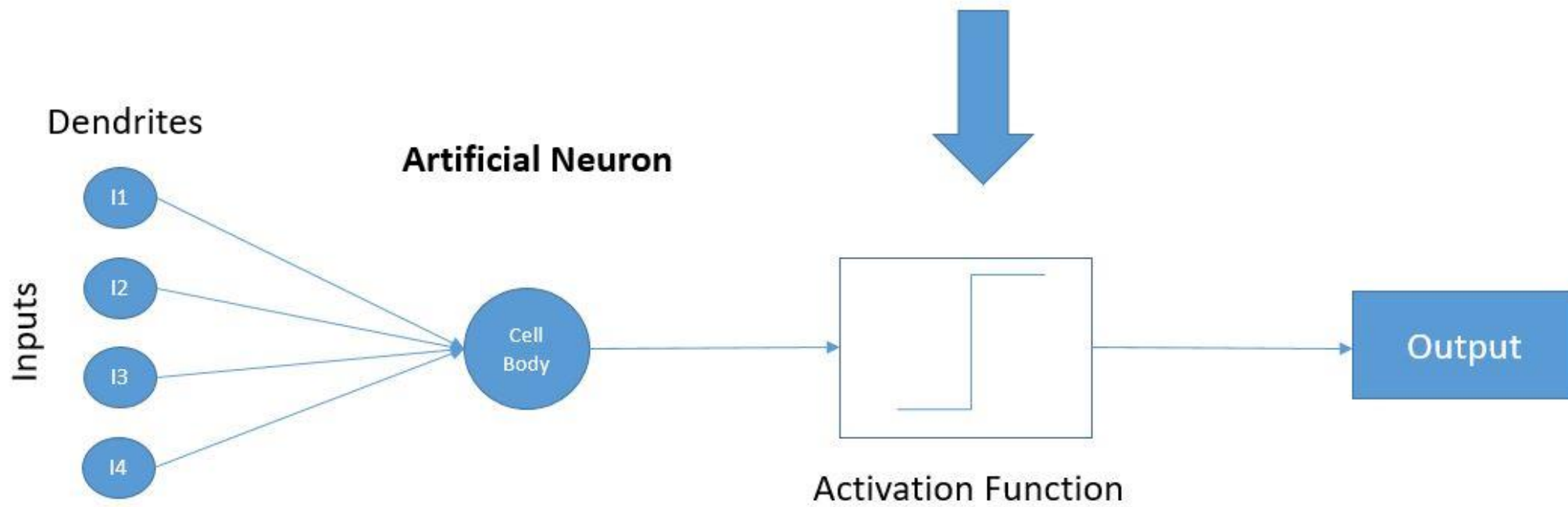
```
plt.scatter(x[:,0],x[:,1], c = y_, cmap='coolwarm')
plt.show()
```



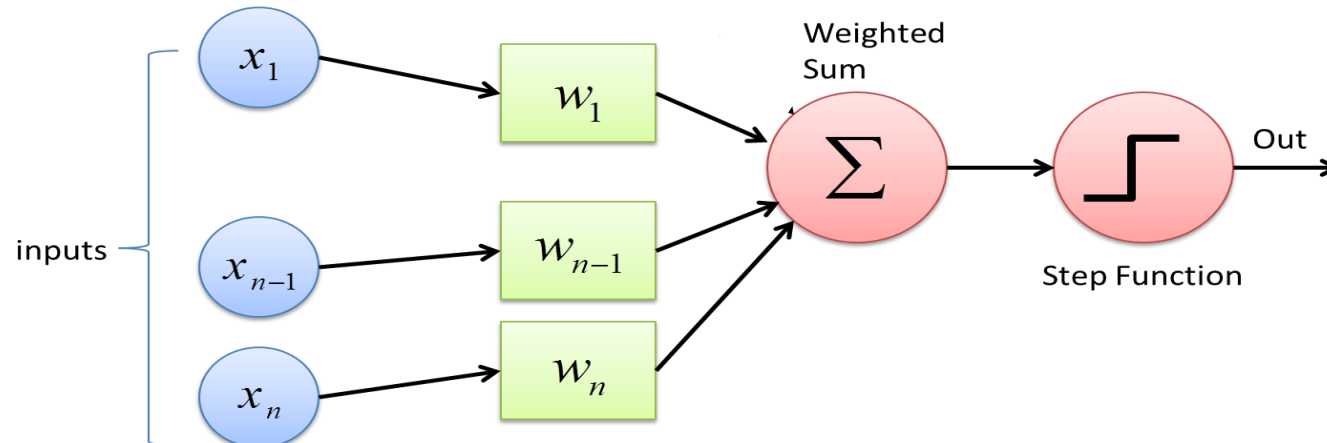
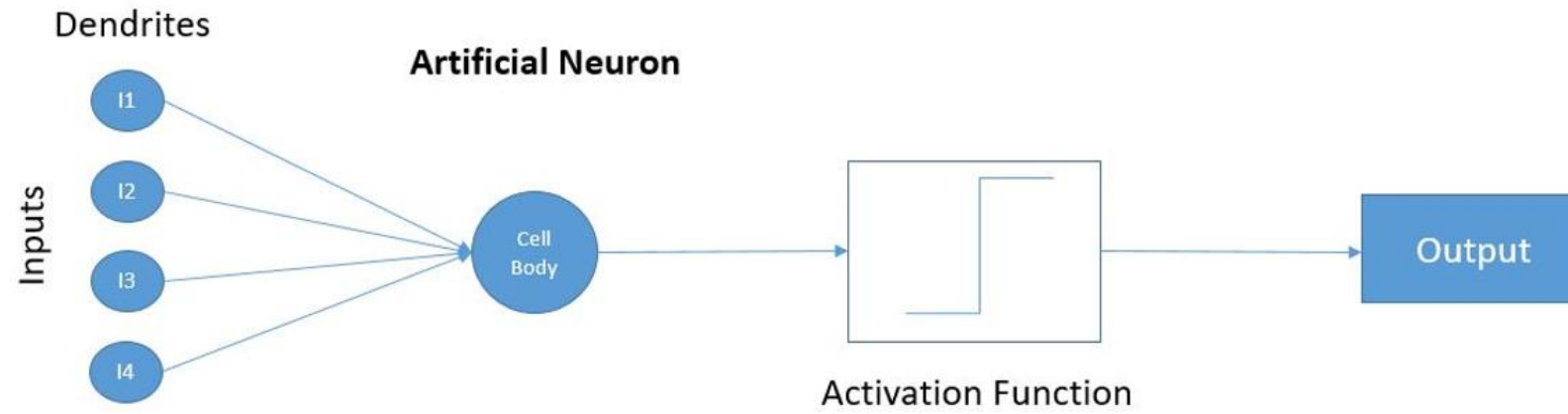
Artificial neural network and deep learning



A neuron

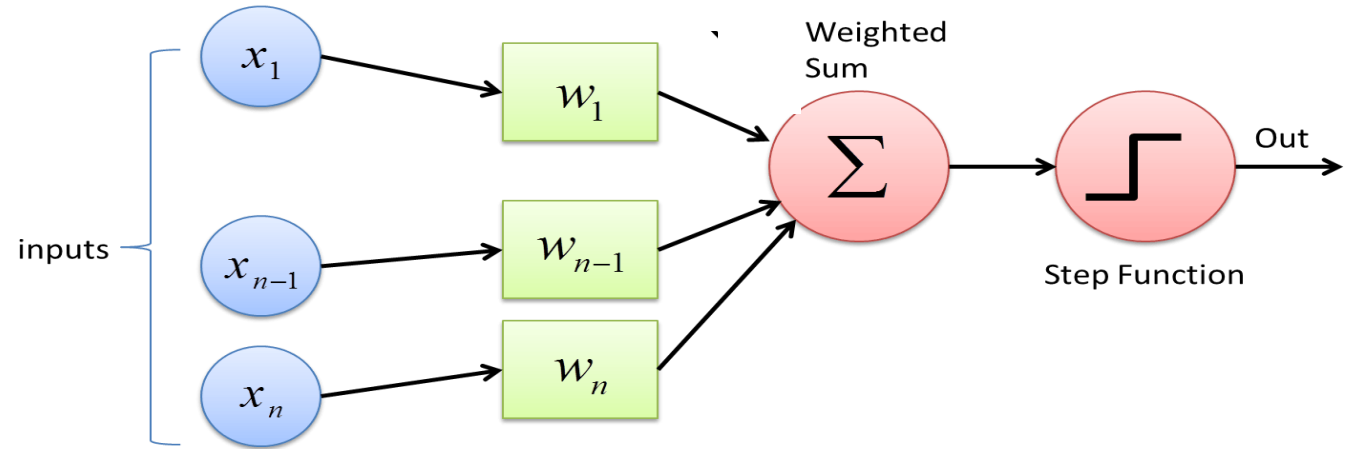


An artificial neuron



Perceptron:
Final model for an
artificial neuron

The Perceptron



We now introduce a simple artificial neural network called Perceptron and explain how to apply it to the *binary classification* task where we refer to our two classes as 1 (positive class) and -1 (negative class). Given a certain input $x = (x_1, x_2, \dots, x_m)$, and a corresponding weights (w_1, w_2, \dots, w_m) , we predict x is in class 1 if the *net input*

$$z = w_1x_1 + w_2x_2 + \dots + w_mx_m$$

is greater than or equal to some threshold θ , and predict -1 otherwise.

The Perceptron

To be more concrete, we define the decision function $\phi(z)$ where

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq \theta \\ -1 & \text{otherwise} \end{cases}$$

We use $\phi(z)$ to help us classify a data point $x = (x_1, x_2, \dots, x_m)$ as follows:

we compute the *net input* $z = w_1x_1 + w_2x_2 + \dots + w_mx_m$ and classify x into the class $\hat{y} = \phi(z) \in \{1, -1\}$.

To train a set of n data points

$$\left\{ x^{(i)} = [x_1^{(i)}, x_2^{(i)}, \dots, x_m^{(i)}] \text{ for } 1 \leq i \leq n \right\},$$

with the set of their class label

$$\left\{ y^{(i)} \in \{1, -1\} \text{ for } 1 \leq i \leq n \right\},$$

we need to decide the best weights w_1, w_2, \dots, w_m and threshold θ such that our prediction

$$\left\{ \hat{y}^{(i)} = \phi(w_1 x_1^{(i)} + w_2 x_2^{(i)} + \dots + w_m x_m^{(i)}) \text{ for } 1 \leq i \leq n \right\}$$

has the smallest number of mis-classifications $y^{(i)} \neq \hat{y}^{(i)}$, or equivalently, the sum

$$\sum_{1 \leq i \leq n} |y^{(i)} - \hat{y}^{(i)}|$$

is as small as possible.

The Perceptron

A simple trick for simplification. Instead of determining two sets of values, namely the weights w_i 's and the threshold θ , we can redefine the net input from

$$z = w_1x_1 + w_2x_2 + \cdots + w_mx_m$$

to

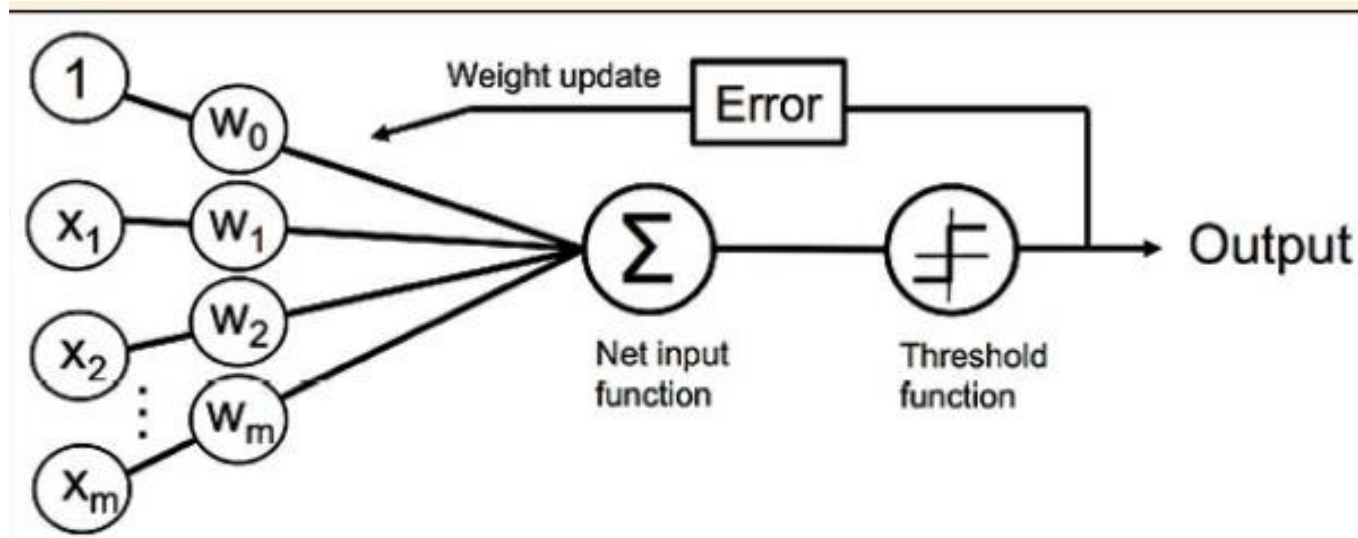
$$z = w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_mx_m$$

where $w_0 = -\theta$ and $x_0 = 1$. Then, we classify $x = (x_1, x_2, \dots, x_m)$ into the class 1 if $z \geq 0$, and -1 otherwise, i.e., we classify x into class $\phi(z)$ where ϕ is redefined as follows:

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

The weight $w_0 = -\theta$ is usually called the *bias unit*.

How to train a perceptron?



How to train a perceptron?

The idea:

- Initialize the weights to some small random numbers.
- For each training example $x^{(i)}$:
 - a. Compute the output value $\hat{y}^{(i)}$ (based on the current weights).
 - b. Update the weights $w_j = w_j + \Delta w_j$.

How to train a perceptron?

What we want:

- If $y^{(i)} = \hat{y}^{(i)}$: We don't want any change, i.e., $\Delta w_j = 0$ for all $1 \leq j \leq m$.
- If $y^{(i)} = 1$ and $\hat{y}^{(i)} = -1$: We want that with the new weights w'_j 's, the value net input

$$z' = w'_0 x_0 + w'_1 x_1 + \cdots + w'_m x_m$$

will increase, i.e., $z' > z$ so that we'd have a better chance to get $z' > 0$ and thus $\hat{y}^{(i)} = \phi(z')$ becomes 1.

- If $y^{(i)} = -1$ and $\hat{y}^{(i)} = 1$: We want that with the new weights w'_j 's, the value net input

$$z' = w'_0 x_0 + w'_1 x_1 + \cdots + w'_m x_m$$

will decrease, i.e., $z' < z$ so that we'd have a better chance to get $z' < 0$ and thus $\hat{y}^{(i)} = \phi(z')$ becomes -1 .

How to train a perceptron?

Rosenblatt's proposed the following rule for Δw_j :

$$\Delta w_j = \eta(y^{(i)} - \hat{y}^{(i)})x_j^{(i)}$$

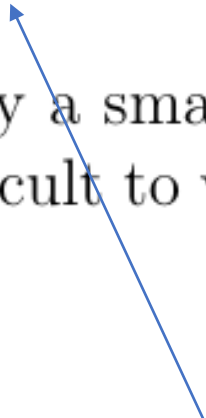
Here, η is the learning rate (typically a small constant between 0.0 and 1.0) for avoiding over-shooting. It is not difficult to verify that Rosenblatt's rule satisfies the above three conditions.

How to train a perceptron?

Rosenblatt's proposed the following rule for Δw_j :

$$\Delta w_j = \eta(y^{(i)} - \hat{y}^{(i)})x_j^{(i)}$$

Here, η is the learning rate (typically a small constant between 0.0 and 1.0) for avoiding over-shooting. It is not difficult to verify that Rosenblatt's rule satisfies the above three conditions.



A very important hyper-parameter, especially for a training data set with million of examples.

Deep learning: TensorFlow and Keras

- TensorFlow is a free software library for machine learning and AI. It can be used across a range of tasks but focuses on training and inference of **deep neural networks**.
- TensorFlow was developed by the Google Brain team.
- Keras is a free software library that provides a Python interface for neural network. It is commonly used as an interface for the TensorFlow library.

Why not Pytorch?

It is "an open source machine learning framework that accelerates the path from research prototyping to production deployment."^[22]

[22] <https://pytorch.org/>

Why not Pytorch?



Written by [Vihar Kurama](#)

PyTorch vs. TensorFlow: My Recommendation

TensorFlow is a very powerful and mature deep learning library with strong visualization capabilities and several options to use for high-level model development. It has production-ready deployment options and support for mobile platforms. PyTorch, on the other hand, is still a relatively young framework with stronger community movement and it's more Python-friendly.

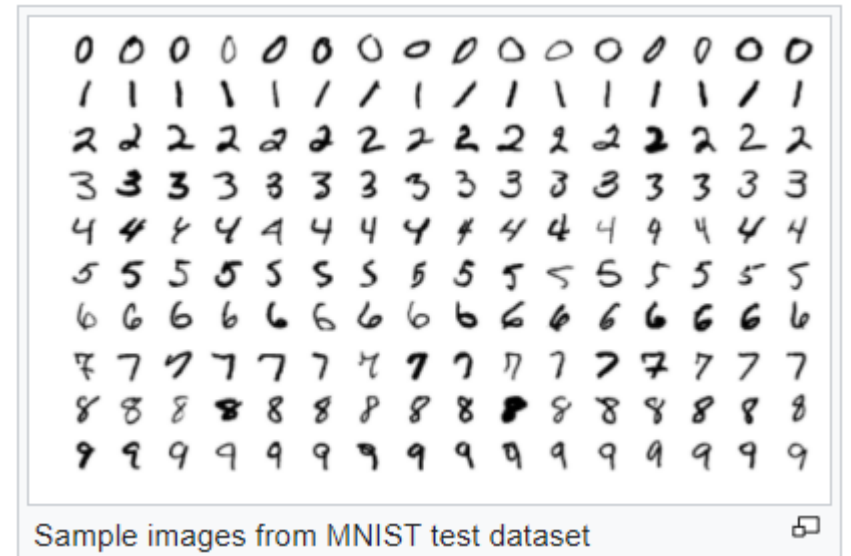
What I would recommend is if you want to make things faster and build AI-related products, TensorFlow is a good choice. PyTorch is mostly recommended for research-oriented developers as it supports fast and dynamic training.

Example: classifying digits

Deep Learning: Train a neural network for digit classification

The MNIST (Modified National Institute of Standards and Technology) dataset for training.

- It is a large dataset of handwritten digits created in 1998.
- Over the years, researchers used this dataset to test the accuracy of different ML methods.



Type	Classifier	Error rate (%)
Linear classifier	Pairwise linear classifier	7.6 ^[10]
Decision stream with Extremely randomized trees	Single model (depth > 400 levels)	2.7 ^[23]
K-Nearest Neighbors	K-NN with non-linear deformation (P2DHMDM)	0.52 ^[24]
Boosted Stumps	Product of stumps on Haar features	0.87 ^[25]
Non-linear classifier	40 PCA + quadratic classifier	3.3 ^[10]
Random Forest	Fast Unified Random Forests for Survival, Regression, and Cla	2.8 ^[27]
Support-vector machine (SVM)	Virtual SVM, deg-9 poly, 2-pixel jittered	0.56 ^[28]
Deep neural network (DNN)	2-layer 784-800-10	1.6 ^[29]
Deep neural network	2-layer 784-800-10	0.7 ^[29]
Deep neural network	6-layer 784-2500-2000-1500-1000-500-10	0.35 ^[30]
Convolutional neural network (CNN)	6-layer 784-40-80-500-1000-2000-10	0.31 ^[31]
Convolutional neural network	6-layer 784-50-100-500-1000-10-10	0.27 ^[32]
Convolutional neural network (CNN)	13-layer 64-128(5x)-256(3x)-512-2048-256-256-10	0.25 ^[17]
Convolutional neural network	Committee of 35 CNNs, 1-20-P-40-P-150-10	0.23 ^[12]
Convolutional neural network	Committee of 5 CNNs, 6-layer 784-50-100-500-1000-10-10	0.21 ^{[19][20]}
Random Multimodel Deep Learning (RMDL)	10 NN-10 RNN - 10 CNN	0.18 ^[22]
Convolutional neural network	Committee of 20 CNNS with Squeeze-and-Excitation Networks	0.17 ^[34]

Deep Learning: Train a neural network for digit classification

```
[1] import numpy as np
import matplotlib.pyplot as plt
from keras import models
from keras import layers
from keras.utils import to_categorical
from keras.datasets import mnist
```

```
[2] (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images.shape, train_labels.shape, test_images.shape, test_labels.shape
```

```
[3] train_images = train_images.reshape((60000, 28*28))/255
test_images = test_images.reshape((10000, 28*28))/255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

```
[4] network = models.Sequential()
network.add(layers.Dense(512, activation = 'relu', input_shape=(28*28,)))
network.add(layers.Dense(10, activation='softmax'))
network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[5] network.fit(train_images, train_labels, epochs=10, batch_size=128)
test_loss, test_acc = network.evaluate(test_images, test_labels)
```

Deep Learning: Train a neural network for digit classification

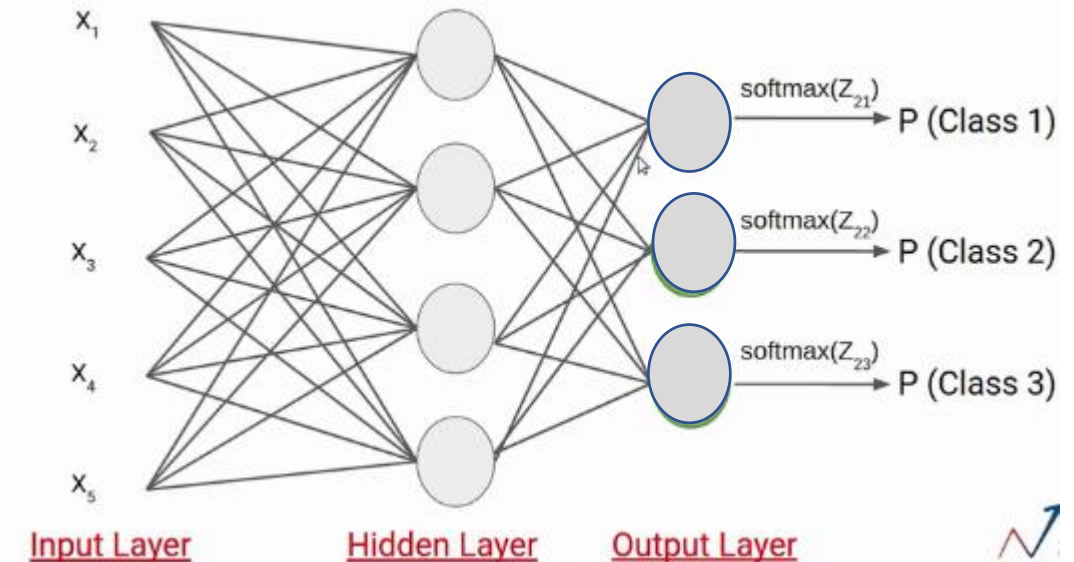
```
[1] import numpy as np
import matplotlib.pyplot as plt
from keras import models
from keras import layers
from keras.utils import to_categorical
from keras.datasets import mnist
```

```
[2] (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images.shape, train_labels.shape, test_images.shape, test_labels.shape
```

```
[3] train_images = train_images.reshape((60000, 28*28))/255
test_images = test_images.reshape((10000, 28*28))/255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

```
[4] network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28*28,)))
network.add(layers.Dense(10, activation='softmax'))
network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[5] network.fit(train_images, train_labels, epochs=10, batch_size=128)
test_loss, test_acc = network.evaluate(test_images, test_labels)
```



Deep Learning: Train a neural network for digit classification

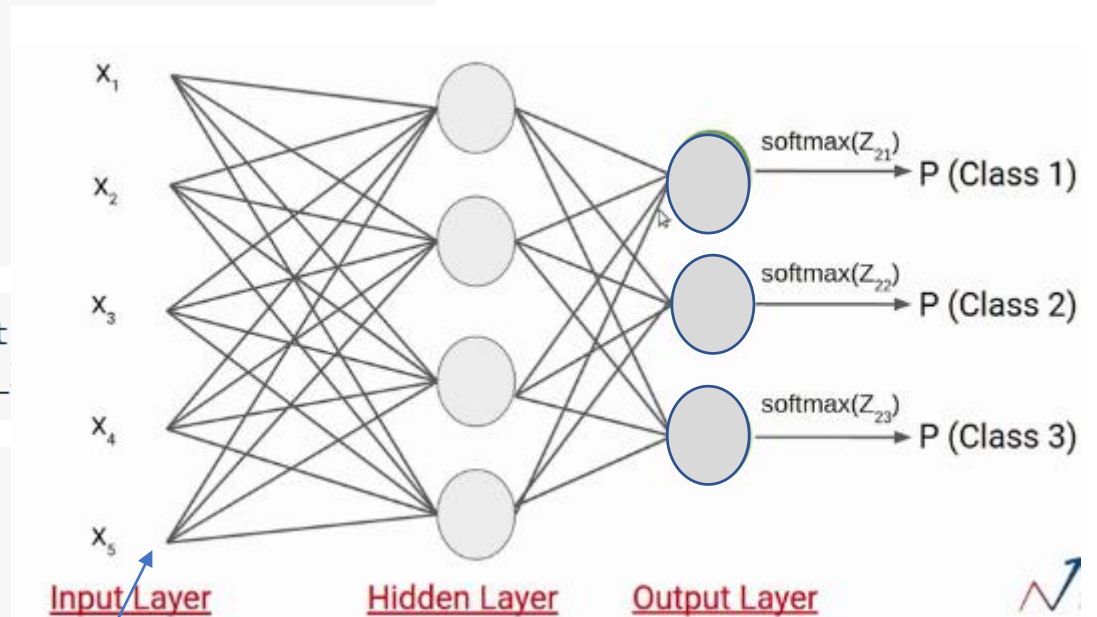
```
[1] import numpy as np
import matplotlib.pyplot as plt
from keras import models
from keras import layers
from keras.utils.np_utils import to_categorical
from keras.datasets import mnist
```

```
[2] (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images.shape, train_labels.shape, test_images.shape, test_labels.shape
```

```
[3] train_images = train_images.reshape((60000, 28*28))/255
test_images = test_images.reshape((10000, 28*28))/255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

```
[4] network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28*28,)))
network.add(layers.Dense(10, activation='softmax'))
network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[5] network.fit(train_images, train_labels, epochs=10, batch_size=128)
test_loss, test_acc = network.evaluate(test_images, test_labels)
```



Deep Learning: Train a neural network for digit classification

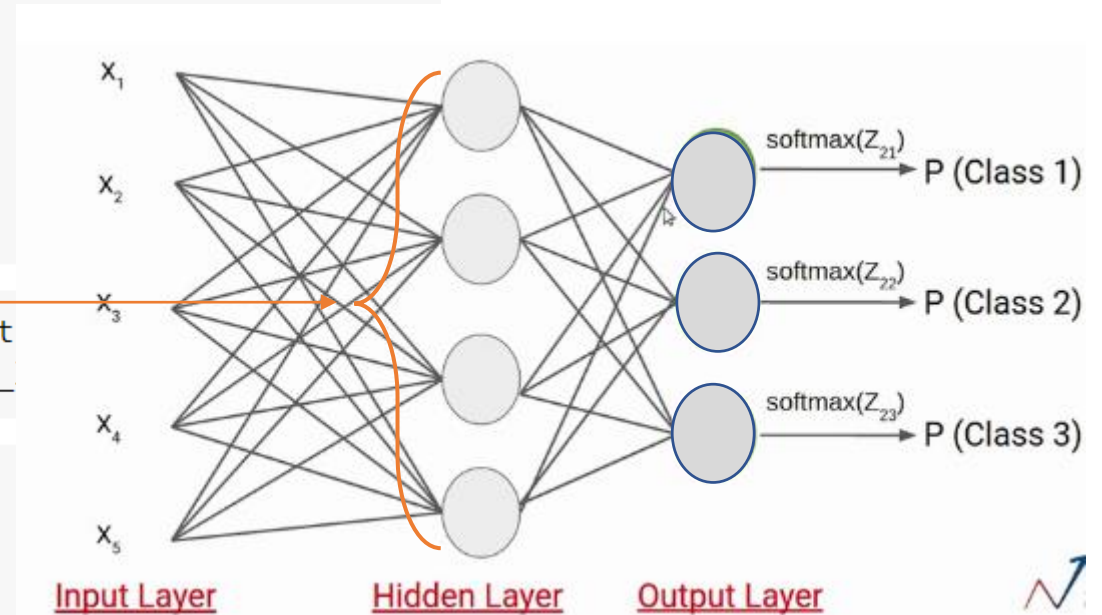
```
[1] import numpy as np
import matplotlib.pyplot as plt
from keras import models
from keras import layers
from keras.utils.np_utils import to_categorical
from keras.datasets import mnist
```

```
[2] (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images.shape, train_labels.shape, test_images.shape, test_labels.shape
```

```
[3] train_images = train_images.reshape((60000, 28*28))/255
test_images = test_images.reshape((10000, 28*28))/255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

```
[4] network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28*28,)))
network.add(layers.Dense(10, activation='softmax'))
network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[5] network.fit(train_images, train_labels, epochs=10, batch_size=128)
test_loss, test_acc = network.evaluate(test_images, test_labels)
```



Deep Learning: Train a neural network for digit classification

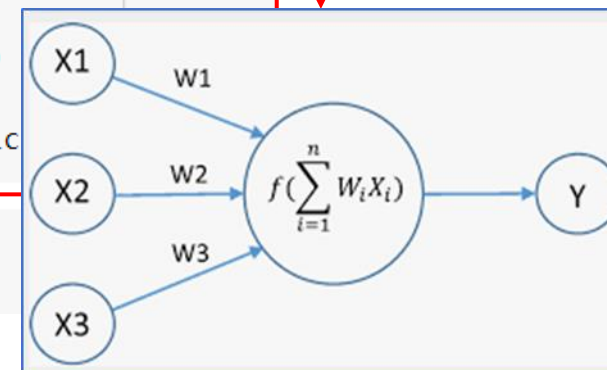
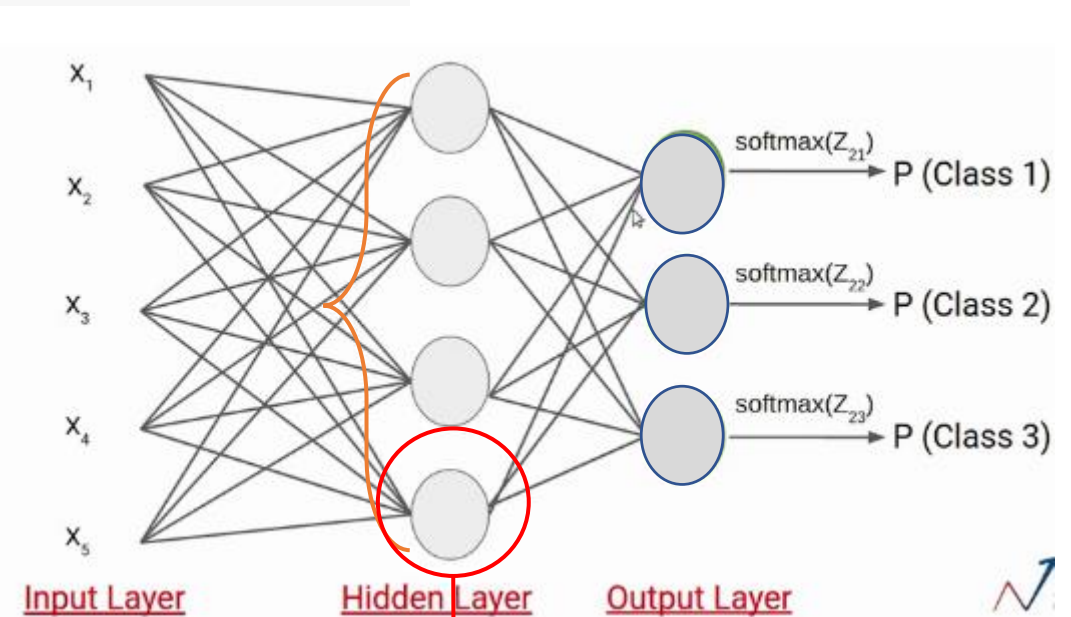
```
[1] import numpy as np
import matplotlib.pyplot as plt
from keras import models
from keras import layers
from keras.utils.np_utils import to_categorical
from keras.datasets import mnist
```

```
[2] (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images.shape, train_labels.shape, test_images.shape, test_labels.shape
```

```
[3] train_images = train_images.reshape((60000, 28*28))/255
test_images = test_images.reshape((10000, 28*28))/255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

```
[4] network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28*28,)))
network.add(layers.Dense(10, activation='softmax'))
network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[5] network.fit(train_images, train_labels, epochs=10, batch_size=128)
test_loss, test_acc = network.evaluate(test_images, test_labels)
```

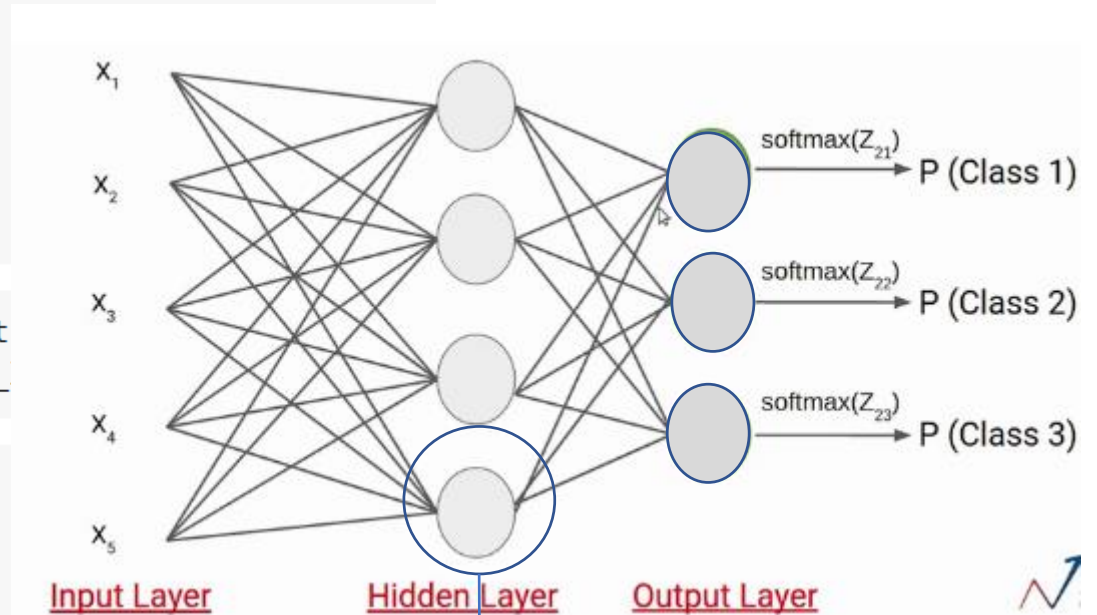


Deep Learning: Train a neural network for digit classification

```
[1] import numpy as np
import matplotlib.pyplot as plt
from keras import models
from keras import layers
from keras.utils.np_utils import to_categorical
from keras.datasets import mnist
```

```
[2] (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images.shape, train_labels.shape, test_images.shape, test_labels.shape
```

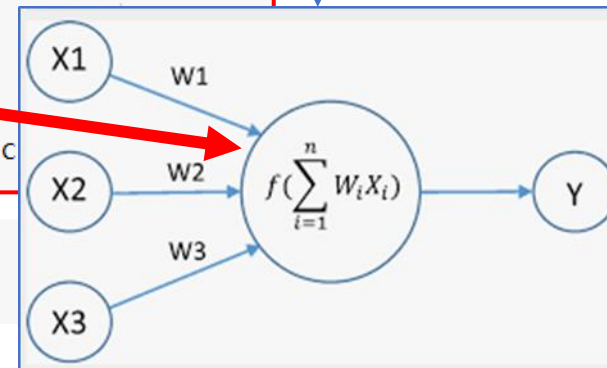
```
[3] train_images = train_images.reshape((60000, 28*28))/255
test_images = test_images.reshape((10000, 28*28))/255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```



```
[4] model = models.Sequential([
    layers.Dense(1000, activation='relu', input_shape=(28*28,)),
    layers.Dense(1000, activation='relu'),
    layers.Dense(10, activation='softmax')])
model.compile(loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[5] model.fit(train_images, train_labels, epochs=10, batch_size=128)
test_loss, test_acc = model.evaluate(test_images, test_labels)
```

'relu', input_shape=(28*28,))
 'softmax',
 'categorical_crossentropy', metric



Deep Learning: Train a neural network for digit classification

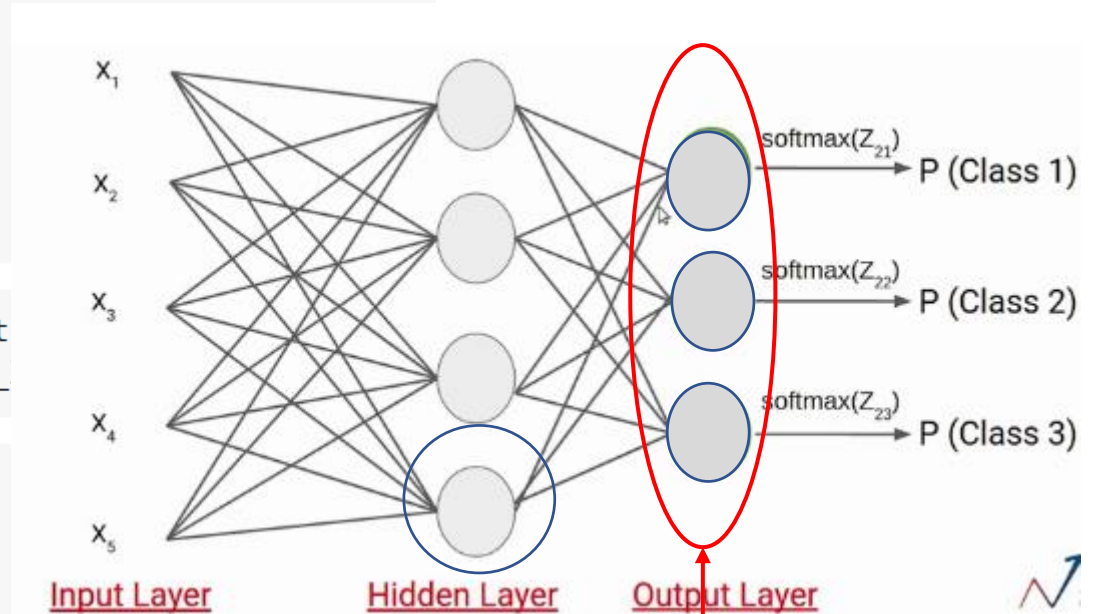
```
[1] import numpy as np
import matplotlib.pyplot as plt
from keras import models
from keras import layers
from keras.utils.np_utils import to_categorical
from keras.datasets import mnist
```

```
[2] (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images.shape, train_labels.shape, test_images.shape, test_labels.shape
```

```
[3] train_images = train_images.reshape((60000, 28*28))/255
test_images = test_images.reshape((10000, 28*28))/255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

```
[4] network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28*28,)))
network.add(layers.Dense(10, activation='softmax'))
network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[5] network.fit(train_images, train_labels, epochs=10, batch_size=128)
test_loss, test_acc = network.evaluate(test_images, test_labels)
```



Deep Learning: Train a neural network for digit classification

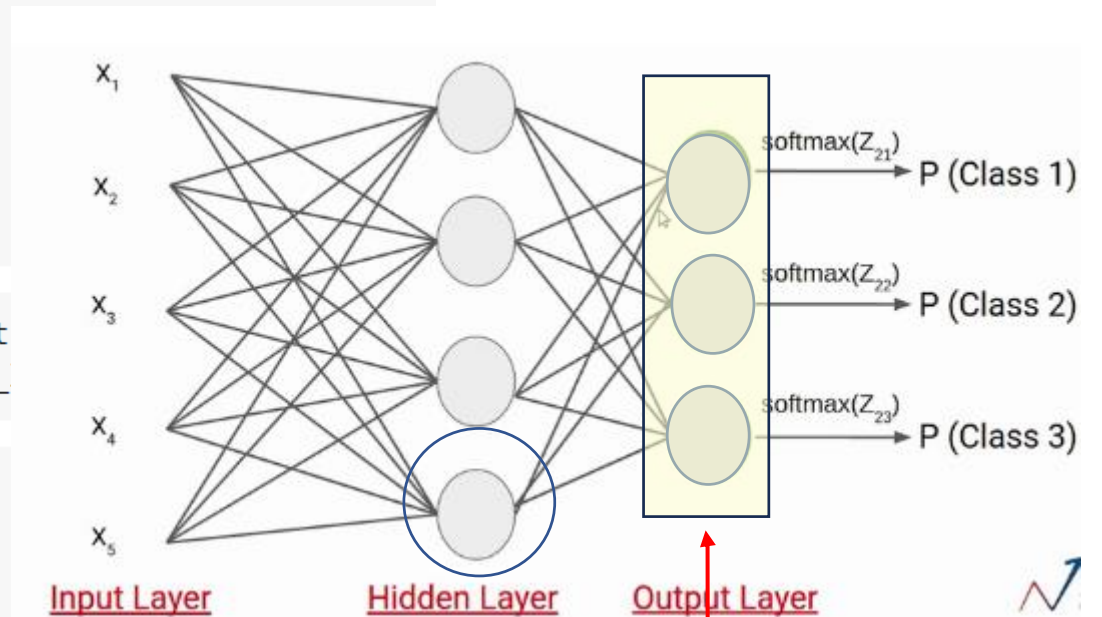
```
[1] import numpy as np
import matplotlib.pyplot as plt
from keras import models
from keras import layers
from keras.utils.np_utils import to_categorical
from keras.datasets import mnist
```

```
[2] (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images.shape, train_labels.shape, test_images.shape, test_labels.shape
```

```
[3] train_images = train_images.reshape((60000, 28*28))/255
test_images = test_images.reshape((10000, 28*28))/255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

```
[4] network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28*28,)))
network.add(layers.Dense(10, activation='softmax'))
network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[5] network.fit(train_images, train_labels, epochs=10, batch_size=128)
test_loss, test_acc = network.evaluate(test_images, test_labels)
```



softmax function

Deep Learning: Train a neural network for digit classification

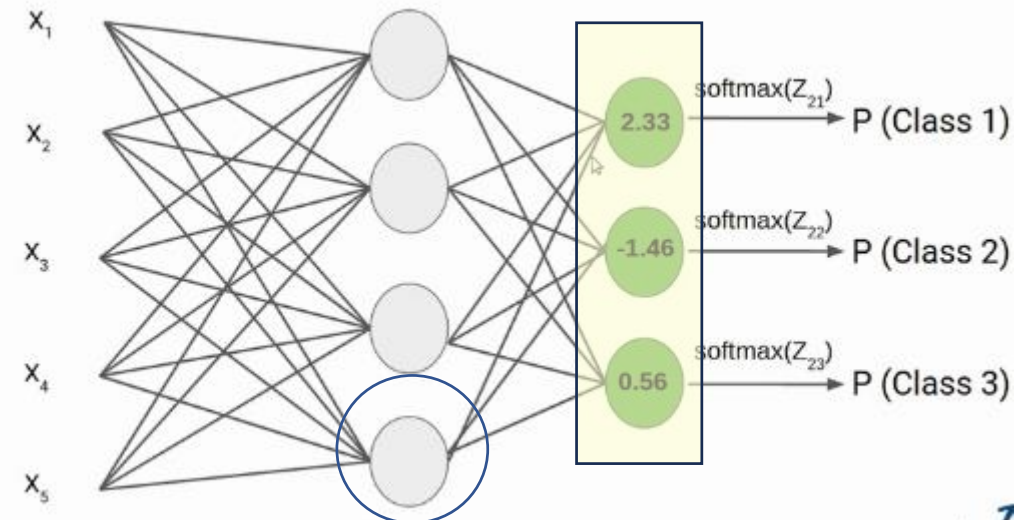
```
[1] import numpy as np
import matplotlib.pyplot as plt
from keras import models
from keras import layers
from keras.utils.np_utils import to_categorical
from keras.datasets import mnist
```

```
[2] (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images.shape, train_labels.shape, test_images.shape, test_labels.shape
```

```
[3] train_images = train_images.reshape((60000, 28*28))/255
test_images = test_images.reshape((10000, 28*28))/255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

```
[4] network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(784,)))
network.add(layers.Dense(10, activation='softmax'))
network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[5] network.fit(train_images, train_labels, epochs=10, batch_size=128)
test_loss, test_acc = network.evaluate(test_images, test_labels)
```



$$2.33 \rightarrow P(\text{Class 1}) = \frac{\exp(2.33)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.83827314$$

$$-1.46 \rightarrow P(\text{Class 2}) = \frac{\exp(-1.46)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.01894129$$

$$0.56 \rightarrow P(\text{Class 3}) = \frac{\exp(0.56)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.14278557$$

Deep Learning: Train a neural network for digit classification

```
[1] import numpy as np
import matplotlib.pyplot as plt
from keras import models
from keras import layers
from keras.utils.np_utils import to_categorical
from keras.datasets import mnist
```

```
[2] (train_images, train_labels),
    (test_images, test_labels) = mnist.load_data()

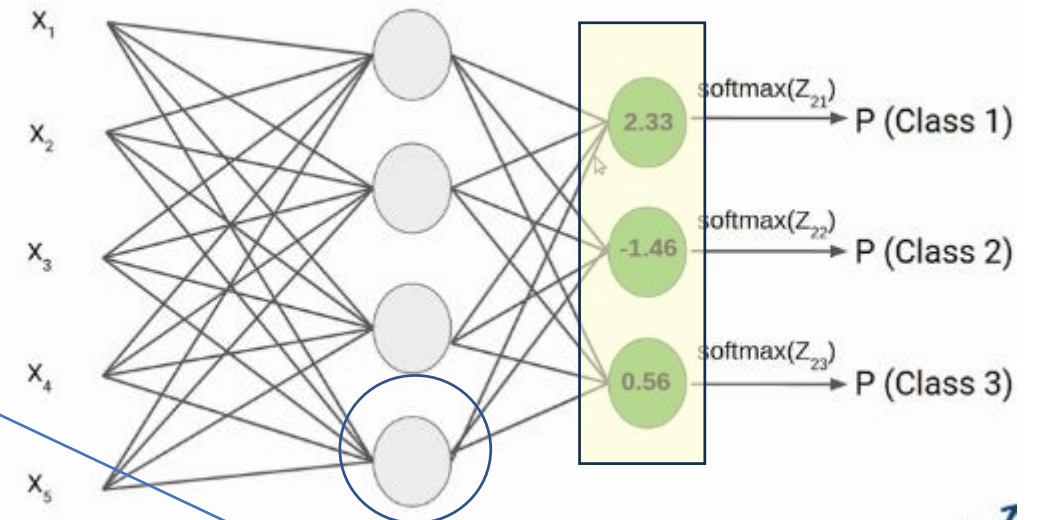
train_images.shape, train_labels.shape
```

The probability that the input is in class 2.

```
[3] train_images = train_images.reshape((60000, 28*28))/255
test_images = test_images.reshape((10000, 28*28))/255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

```
[4] network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(784,)))
network.add(layers.Dense(10, activation='softmax'))
network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[5] network.fit(train_images, train_labels, epochs=10, batch_size=128)
test_loss, test_acc = network.evaluate(test_images, test_labels)
```



$$2.33 \rightarrow P(\text{Class 1}) = \frac{\exp(2.33)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.83827314$$

$$-1.46 \rightarrow P(\text{Class 2}) = \frac{\exp(-1.46)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.01894129$$

$$0.56 \rightarrow P(\text{Class 3}) = \frac{\exp(0.56)}{\exp(2.33) + \exp(-1.46) + \exp(0.56)} = 0.14278557$$

Deep Learning: Train a neural network for digit classification

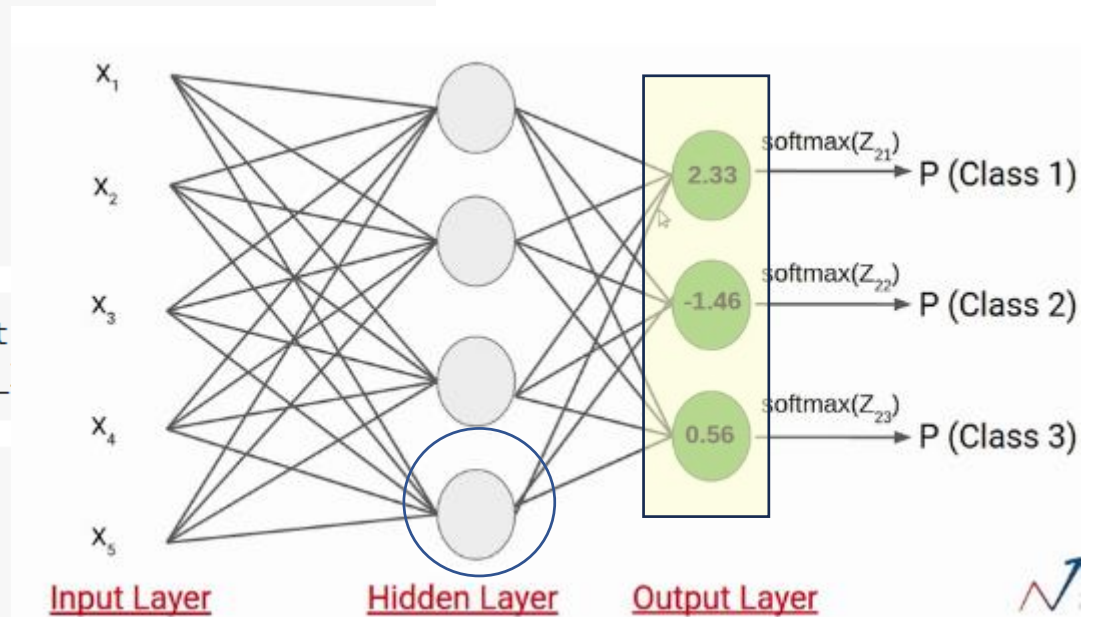
```
[1] import numpy as np
import matplotlib.pyplot as plt
from keras import models
from keras import layers
from keras.utils.np_utils import to_categorical
from keras.datasets import mnist
```

```
[2] (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images.shape, train_labels.shape, test_images.shape, test_labels.shape
```

```
[3] train_images = train_images.reshape((60000, 28*28))/255
test_images = test_images.reshape((10000, 28*28))/255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

```
[4] network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28*28,)))
network.add(layers.Dense(10, activation='softmax'))
network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[5] network.fit(train_images, train_labels, epochs=10, batch_size=128)
test_loss, test_acc = network.evaluate(test_images, test_labels)
```



Deep Learning: Train a neural network for digit classification

```
[1] import numpy as np
import matplotlib.pyplot as plt
from keras import models
from keras import layers
from keras.utils.np_utils import to_categorical
from keras.datasets import mnist
```

```
[2] (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images.shape, train_labels.shape, test_images.shape, test_labels.shape
```

```
[3] train_images = train_images.reshape((60000, 28*28))/255
test_images = test_images.reshape((10000, 28*28))/255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

```
[4] network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28*28,)))
network.add(layers.Dense(10, activation='softmax'))
network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[5] network.fit(train_images, train_labels, epochs=10, batch_size=128)
test_loss, test_acc = network.evaluate(test_images, test_labels)
```

x_1

x_2

x_3

x_4

x_5

Input

neuralthreads - Medium

Categorical cross-entropy loss -

$$CE = - \sum_{i=1}^{i=N} y_{true_i} \cdot \log(y_{pred_i})$$

$$CE = - \sum_{i=1}^{i=N} y_i \cdot \log(\hat{y}_i)$$

$$\Rightarrow CE = -[y_1 \cdot \log(\hat{y}_1) + y_2 \cdot \log(\hat{y}_2) + y_3 \cdot \log(\hat{y}_3)]$$

?????

You don't need to remember it. All you need to know is the optimizer will set the weights appropriately to make CE as small as possible (over the training data set).

Deep Learning: Train a neural network for digit classification

```
[1] import numpy as np
import matplotlib.pyplot as plt
from keras import models
from keras import layers
from keras.utils.np_utils import to_categorical
from keras.datasets import mnist
```

```
[2] (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images.shape, train_labels.shape, test_images.shape, test_labels.shape
```

```
[3] train_images = train_images.reshape((60000, 28*28))/255
test_images = test_images.reshape((10000, 28*28))/255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

```
[4] network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28*28,)))
network.add(layers.Dense(10, activation='softmax'))
network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[5] network.fit(train_images, train_labels, epochs=10, batch_size=128)
test_loss, test_acc = network.evaluate(test_images, test_labels)
```

The algorithm we choose to minimize the loss functions. There are many choices:

- SGD
- RMSprop
- Adam
- AdamW
- Adadelta
- Adagrad
- Adamax
- Adafactor
- Nadam
- Ftrl
- Lion
- Loss Scale Optimizer

Deep Learning: Train a neural network for digit classification

```
[1] import numpy as np
import matplotlib.pyplot as plt
from keras import models
from keras import layers
from keras.utils.np_utils import to_categorical
from keras.datasets import mnist
```

```
[2] (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images.shape, train_labels.shape, test_images.shape, test_labels.shape
```

```
[3] train_images = train_images.reshape((60000, 28*28))/255
test_images = test_images.reshape((10000, 28*28))/255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

Behaviour of different optimizer: [CS231n Convolutional Neural Networks for Visual Recognition](#)

```
[4] network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28*28,)))
network.add(layers.Dense(10, activation='softmax'))
network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[5] network.fit(train_images, train_labels, epochs=10, batch_size=128)
test_loss, test_acc = network.evaluate(test_images, test_labels)
```

The algorithm we choose to minimize the loss functions. There are many choices:

- SGD
- RMSprop
- Adam
- AdamW
- Adadelta
- Adagrad
- Nadam
- Ftrl
- Lion
- Loss Scale Optimizer

Deep Learning: Train a neural network for digit classification

```
[1] import numpy as np
import matplotlib.pyplot as plt
from keras import models
from keras import layers
from keras.utils.np_utils import to_categorical
from keras.datasets import mnist
```

```
[2] (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images.shape, train_labels.shape, test_images.shape, test_labels.shape
```

```
[3] train_images = train_images.reshape((60000, 28*28))/255
test_images = test_images.reshape((10000, 28*28))/255
```

One epoch is one complete pass of the training dataset through the algorithm

```
[4] network = models.Sequential()
network.add(layers.Dense(512, activation = 'relu', input_shape=(28*28,)))
network.add(layers.Dense(10, activation='softmax'))
network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[5] network.fit(train_images, train_labels, epochs=10, batch_size=128)
test_loss, test_acc = network.evaluate(test_images, test_labels)
```

Deep Learning: Train a neural network for digit classification

```
[1] import numpy as np
import matplotlib.pyplot as plt
from keras import models
from keras import layers
from keras.utils.np_utils import to_categorical
from keras.datasets import mnist
```

```
[2] (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images.shape, train_labels.shape, test_images.shape, test_labels.shape
```

```
[3] train_images = train_images.reshape((60000, 28*28))/255
test_images = test_images.reshape((10000, 28*28))/255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

```
[4] network = models.Sequential()
network.add(layers.Dense(512, activation = 'relu', input_shape=(28*28,)))
network.add(layers.Dense(10, activation='softmax'))
network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[5] network.fit(train_images, train_labels, epochs=10, batch_size=128)
test_loss, test_acc = network.evaluate(test_images, test_labels)
```

The number of data points used for an update of the weights

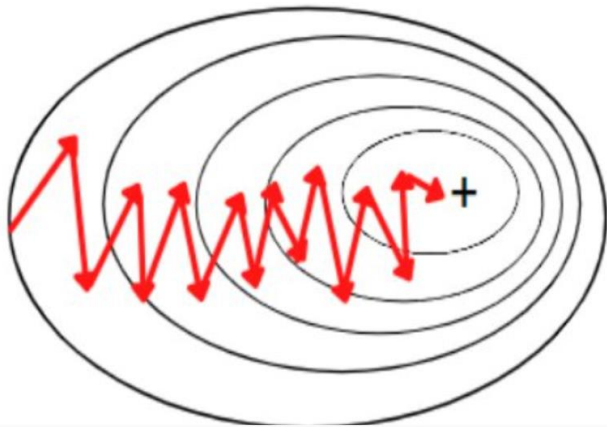
Deep Learning: Train a neural network for digit classification

```
[1] import numpy as np
import matplotlib.pyplot as plt
from keras import models
from keras import layers
from keras.utils.np_utils import to_categorical
from keras.datasets import mnist
```

batch_size = 1: Each update is fast,
but the path to the minimum is
not smooth: there are ups and down

```
[2] train_images, test_images, train_labels, test_labels = mnist.load_data()
train_images = train_images.reshape((train_images.shape[0], test_images.shape[0],
```

```
[3] (train_images[0:10000, 28*28))/255
(train_images[0:100, 28*28))/255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```



```
[4] model = models.Sequential([
    layers.Dense(128, activation='relu', input_shape=(28*28,)),
    layers.Dense(10, activation='softmax')
])
model.compile(loss='categorical_crossentropy', metrics=['accuracy'])
```

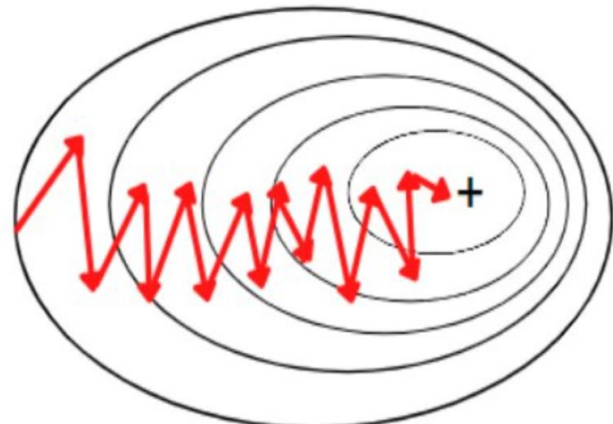
```
[5] network.fit(train_images, train_labels, epochs=10, batch_size=128)
test_loss, test_acc = network.evaluate(test_images, test_labels)
```

Deep Learning: Train a neural network for digit classification

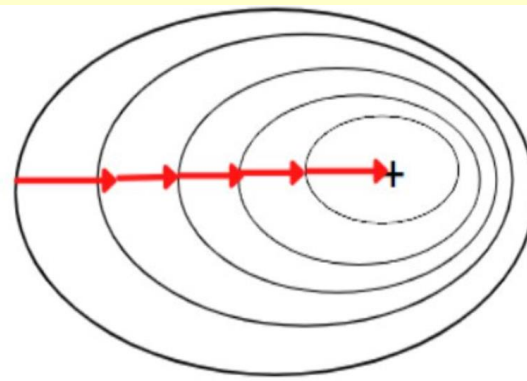
```
[1] import numpy as np
import matplotlib.pyplot as plt
from keras import models
from keras import layers
from keras.utils.np_utils import to_categorical
from keras.datasets import mnist
```

batch_size = 1: Each update is fast,
but the path to the minimum is
not smooth: there are ups and down

batch_size = the size of the whole data
set: Each update is very slow,
but the path to the minimum is
very smooth



```
0000,  
100, 28  
ibels)  
:ls)
```



```
curacy']])
```

```
[5] network.fit(train_images, train_labels, epochs=10, batch_size=128)  
test_loss, test_acc = network.evaluate(test_images, test_labels)
```

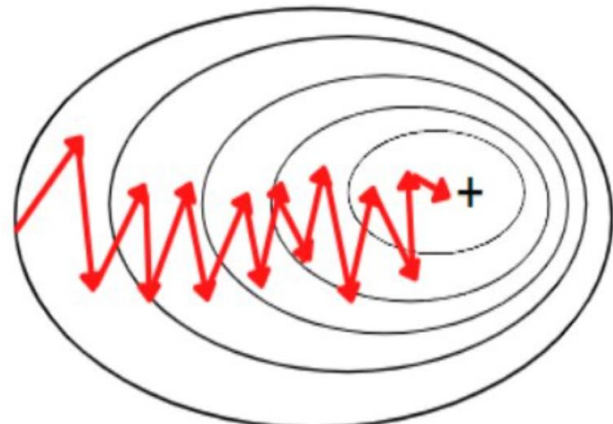
Deep Learning: Train a neural network for digit classification

```
[1] import numpy as np
import matplotlib.pyplot as plt
from keras import models
from keras import layers
from keras.utils.np_utils import to_categorical
from keras.datasets import mnist
```

batch_size = 1: Each update is fast,
but the path to the minimum is
not smooth: there are ups and down

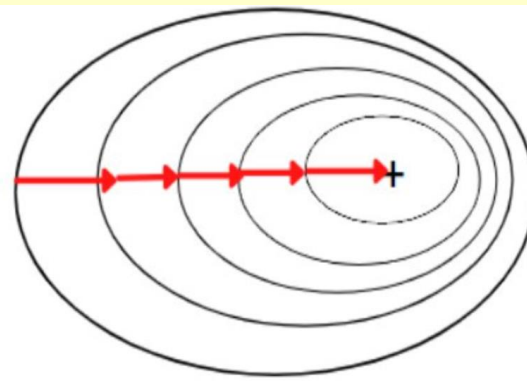
batch_size = the size of the whole data
set: Each update is very slow,
but the path to the minimum is
very smooth

batch_size = somewhere between
1 and dataset's size

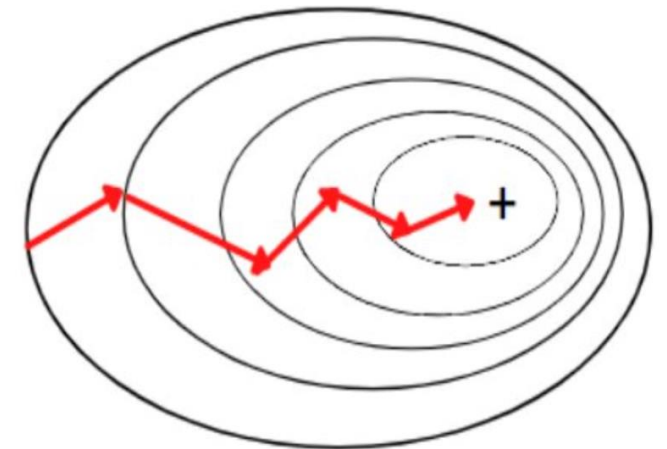


```
00000,
100, 28
ibels)
:ls)
```

```
.on = '
i='soft
ss='ca
```



```
curacy']])
```



```
[5] network.fit(train_images,train_labels, epochs=10, batch_size=128)
test_loss, test_acc = network.evaluate(test_images, test_labels)
```