Unit 4: Machine learning and Deep Learning for Time Series

Generating and Selecting Features for a Time Series

Concerns you should keep in mind when generating time series features.

- Nature of the Time Series : Stationarity, Length of Time series
- Domain Knowledge
- External Considerations

Some Simple and often used time series features are:

- Mean and variance, Maximum and minimum
- Difference between last and first values

Number of local maxima and minima

- Smoothness of the time series, Periodicity and autocorrelation of the time series

Open Source Time Series Feature Generation Libraries

- The ***tsfresh*** Python module: It implement large and general set of features which includes
  - *Descriptive statistics*
  - *Physics-inspired indicators of nonlinearity and complexity*
  - *History-compressing counts*
- The ***Cesium*** time series analysis platform
  - Features that describe the overall distribution of the data values, without regard to its temporal relationships.
  - Features that describe the distribution of the timing of data:
  - Features that describe measures of the periodicity of the behavior within the time series.

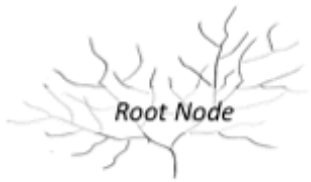
Time Series Classification: Machine learning Methods

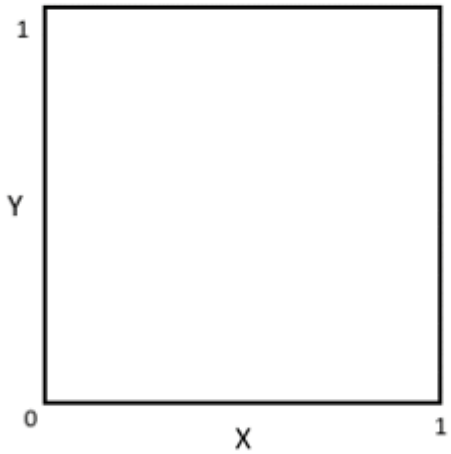
Decision Tree Methods

- Tree-based methods mirror the way humans make decisions: one step at a time, and in a highly nonlinear fashion.

For example : Stock marketing, Medical signals like EEG and ECG

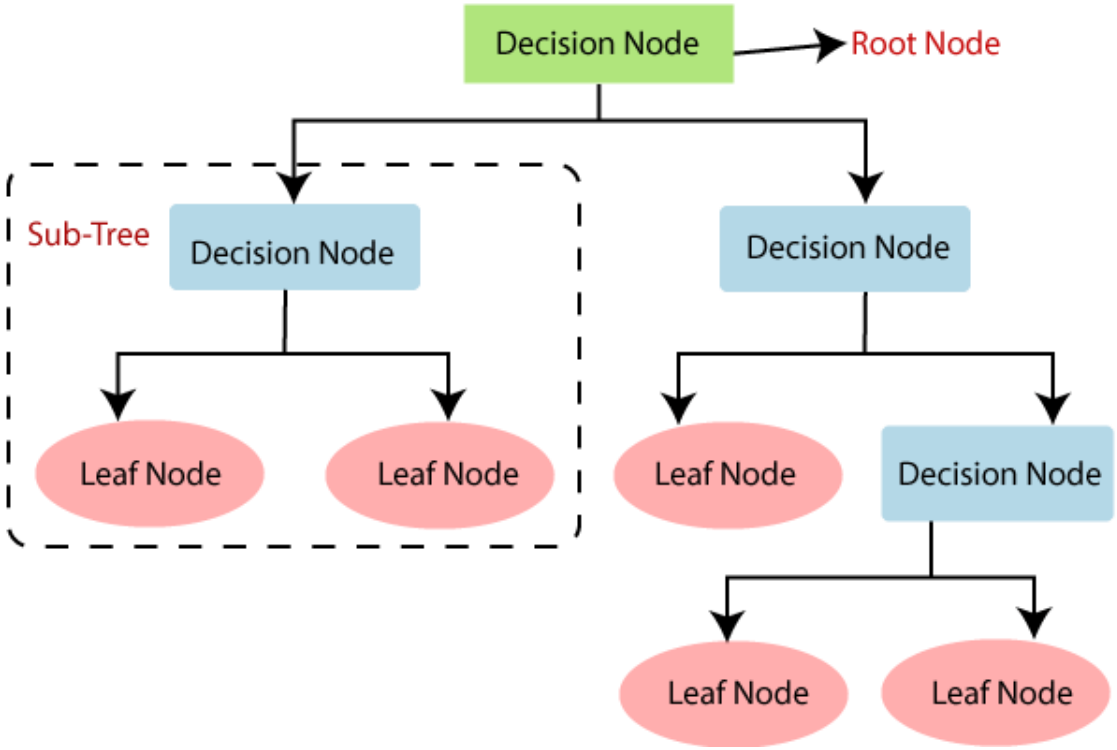
Decision Tree





For more tutorials: annalyzin.wordpress.com

Terminologies in Decision Tree



Root Node: represents entire population and further divided in two/ more homogeneous sets.

Splitting: Process of dividing node in two or more sub-nodes.

Decision Node: When a sub-node splits into further sub-nodes

Leaf / Terminal Node: Nodes do not split.

Pruning: When we remove sub-nodes of a decision node.

Branch/Sub-Tree: A subsection of the entire tree.

Parent and Child Node: A node, which is divided into sub-nodes is called a parent node of sub-nodes whereas sub-nodes are the child of a parent node.

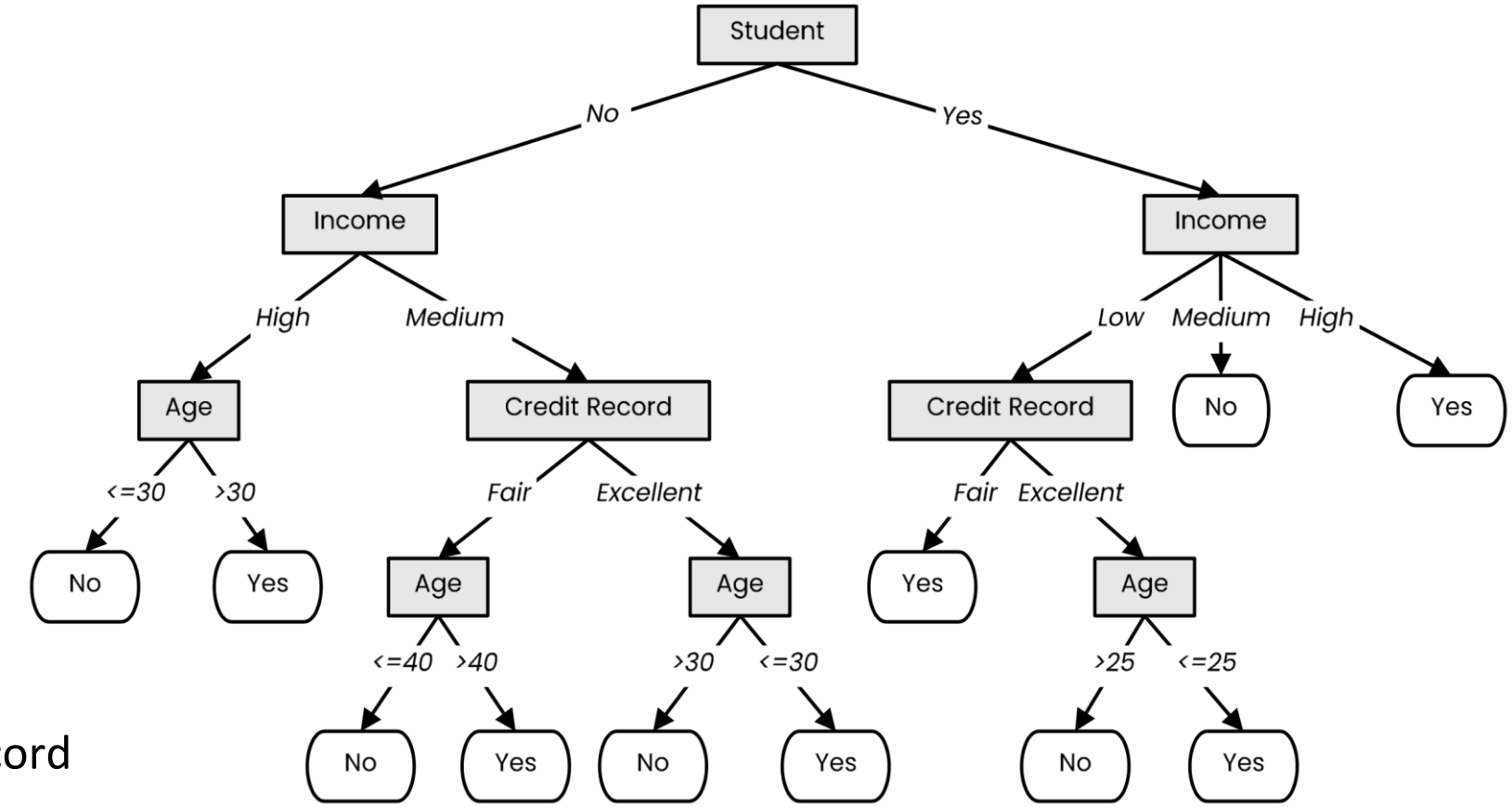
Definition

- A tree-like model that illustrates series of events leading to certain decisions
- Each node represents a test on an attribute and each branch is an outcome of that test

Who to loan?

- Not a student
- 45 years old
- Medium income
- Fair credit record

- Student
- 27 years old
- Low income
- Excellent credit record



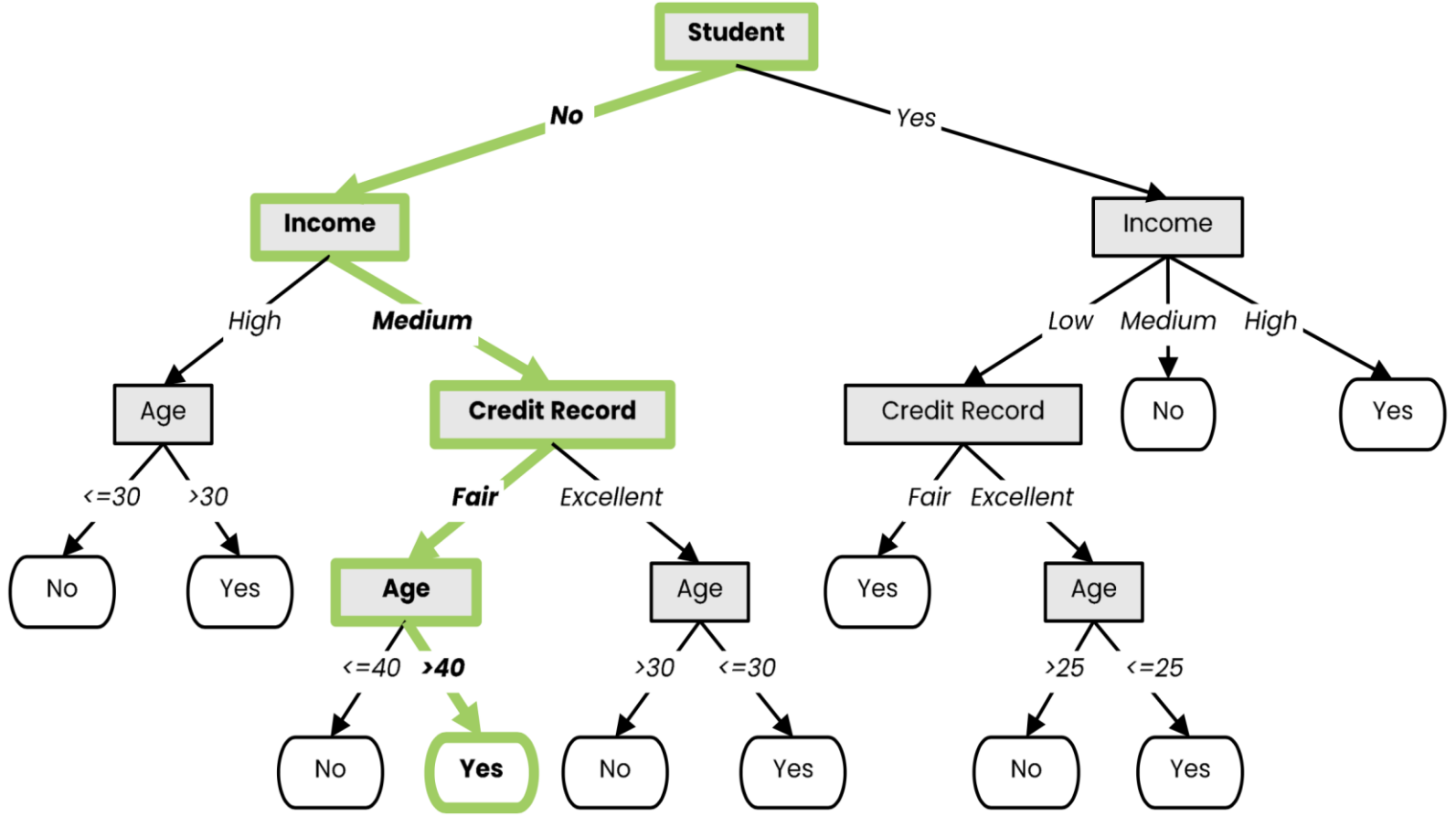
Definition

- A tree-like model that illustrates series of events leading to certain decisions
- Each node represents a test on an attribute and each branch is an outcome of that test

Who to loan?

- Not a student
- 45 years old
- Medium income
- Fair credit record
- Yes

- Student
- 27 years old
- Low income
- Excellent credit record



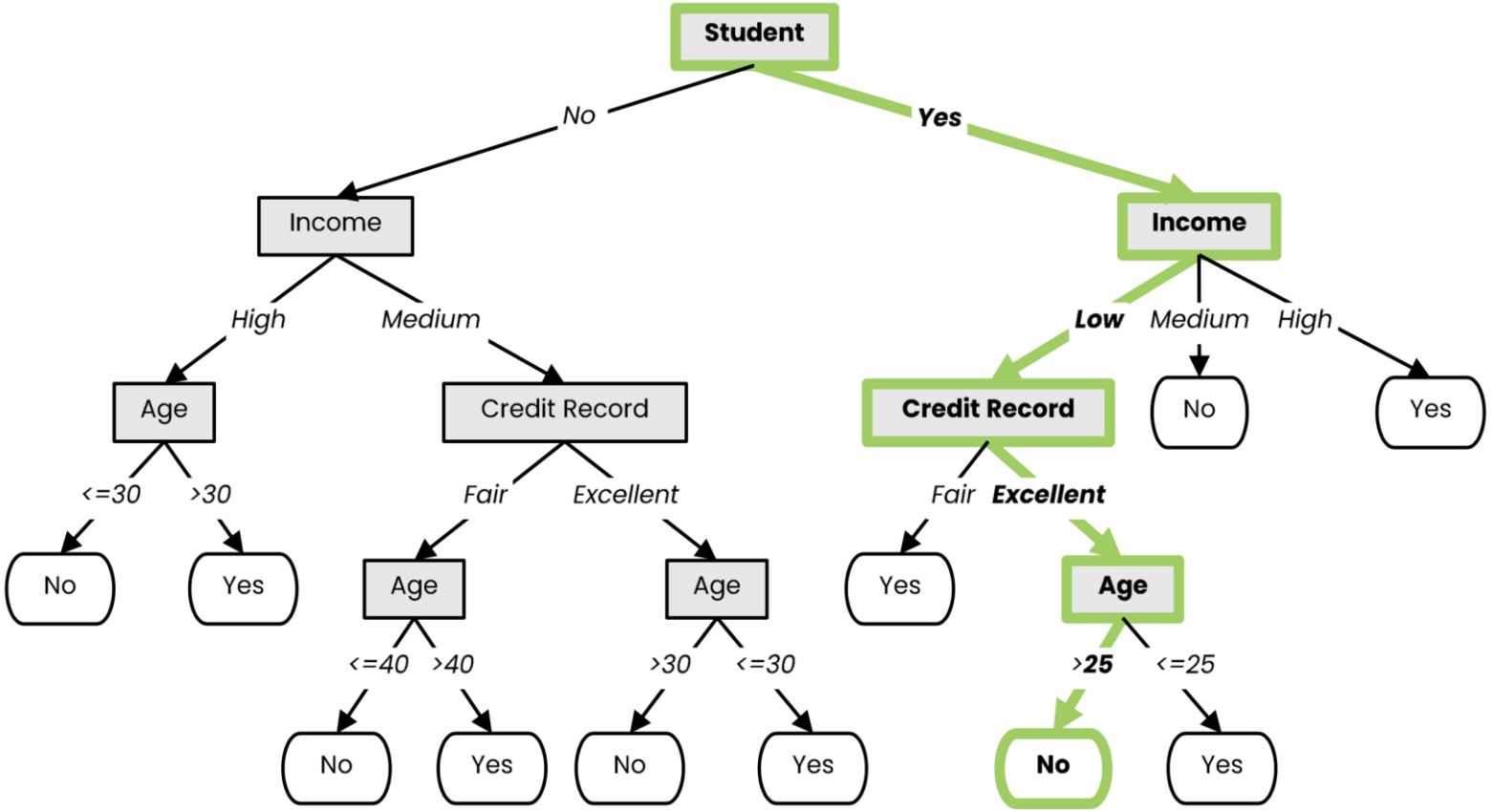
Definition

- A tree-like model that illustrates series of events leading to certain decisions
- Each node represents a test on an attribute and each branch is an outcome of that test

Who to loan?

- Not a student
- 45 years old
- Medium income
- Fair credit record
- Yes

- Student
- 27 years old
- Low income
- Excellent credit record



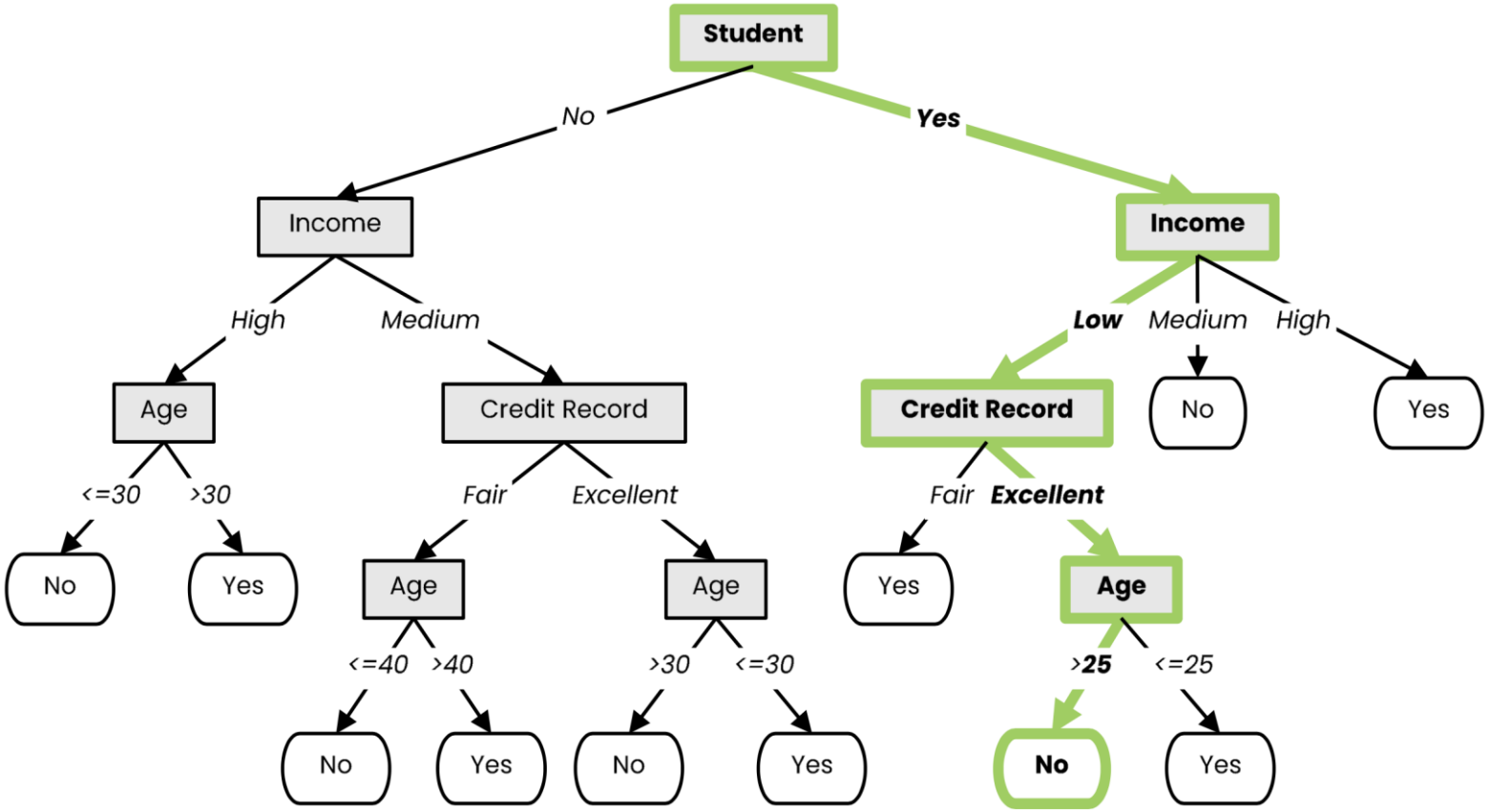
Definition

- A tree-like model that illustrates series of events leading to certain decisions
- Each node represents a test on an attribute and each branch is an outcome of that test

Who to loan?

- Not a student
- 45 years old
- Medium income
- Fair credit record
- Yes

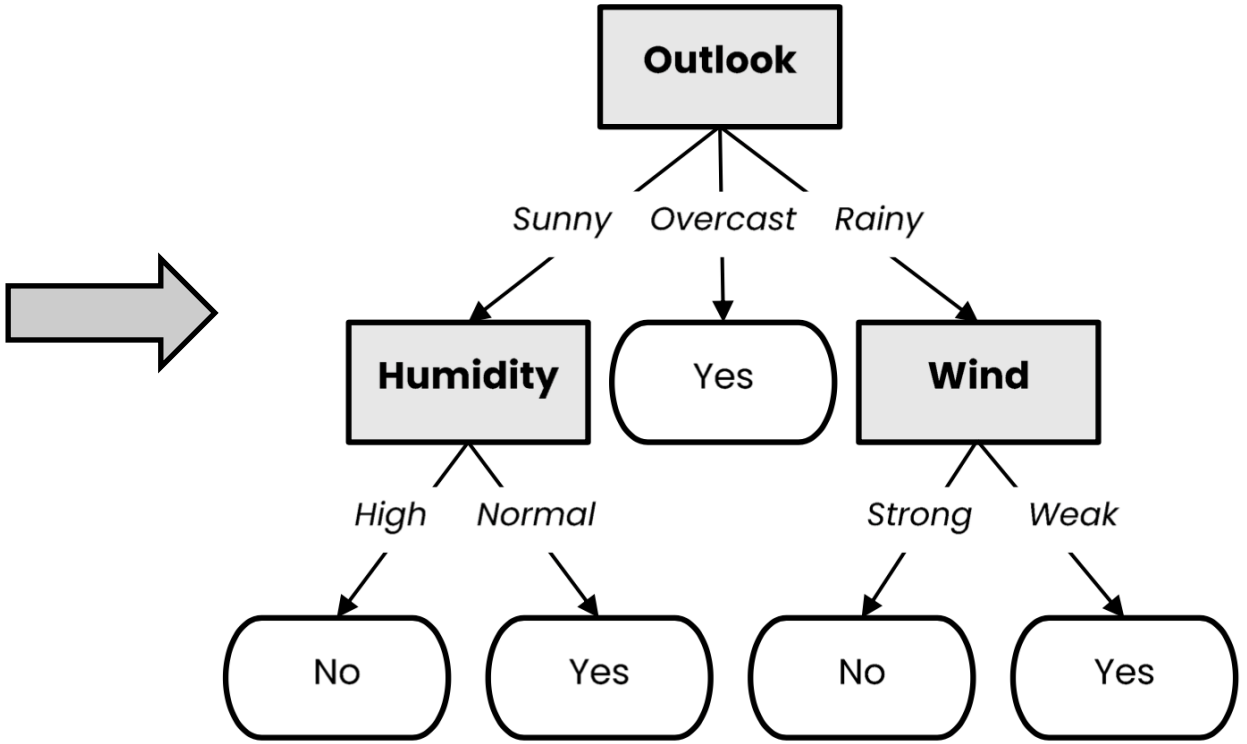
- Student
- 27 years old
- Low income
- Excellent credit record
- No



Decision Tree Learning

- We use labeled data to obtain a suitable decision tree for future predictions
- We want a decision tree that works well on unseen data, while asking as few questions as possible

Outlook	Temperature	Humidity	Wind	Play Tennis?
Sunny	Hot	High	Weak	**No**
Sunny	Hot	High	Strong	**No**
Overcast	Hot	High	Weak	**Yes**
Rainy	Mild	High	Weak	**Yes**
Rainy	Cool	Normal	Weak	**Yes**
Rainy	Cool	Normal	Strong	**No**
Overcast	Cool	Normal	Strong	**Yes**
Sunny	Mild	High	Weak	**No**
Sunny	Cool	Normal	Weak	**Yes**
Rainy	Mild	Normal	Weak	**Yes**
Sunny	Mild	Normal	Strong	**Yes**
Overcast	Mild	High	Strong	**Yes**
Overcast	Hot	Normal	Weak	**Yes**
Rainy	Mild	High	Strong	**No**



Decision Tree Learning

- Basic step: choose an attribute and, based on its values, split the data into smaller sets
  - Recursively repeat this step until we can surely decide the label

Outlook	Temperature	Humidity	Wind	Play Tennis?
Sunny	Hot	High	Weak	**No**
Sunny	Hot	High	Strong	**No**
Overcast	Hot	High	Weak	**Yes**
Rainy	Mild	High	Weak	**Yes**
Rainy	Cool	Normal	Weak	**Yes**
Rainy	Cool	Normal	Strong	**No**
Overcast	Cool	Normal	Strong	**Yes**
Sunny	Mild	High	Weak	**No**
Sunny	Cool	Normal	Weak	**Yes**
Rainy	Mild	Normal	Weak	**Yes**
Sunny	Mild	Normal	Strong	**Yes**
Overcast	Mild	High	Strong	**Yes**
Overcast	Hot	Normal	Weak	**Yes**
Rainy	Mild	High	Strong	**No**

Decision Tree Learning

- Basic step: choose an attribute and, based on its values, split the data into smaller sets
  - Recursively repeat this step until we can surely decide the label

Outlook = *Sunny*

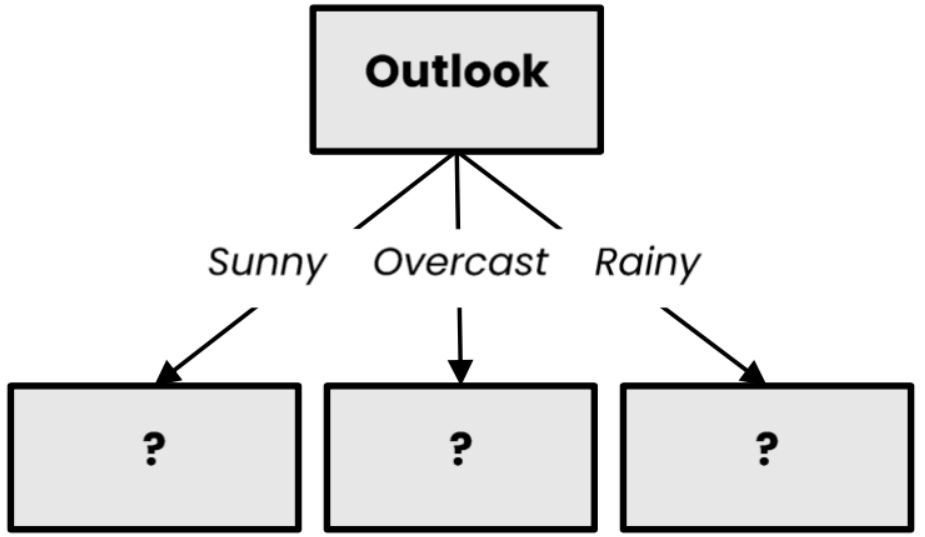
Temperature	Humidity	Wind	Play Tennis?
Hot	High	Weak	**No**
Hot	High	Strong	**No**
Mild	High	Weak	**No**
Cool	Normal	Weak	**Yes**
Mild	Normal	Strong	**Yes**

Outlook = *Overcast*

Temperature	Humidity	Wind	Play Tennis?
Hot	High	Weak	**Yes**
Cool	Normal	Strong	**Yes**
Mild	High	Strong	**Yes**
Hot	Normal	Weak	**Yes**

Outlook = *Rainy*

Temperature	Humidity	Wind	Play Tennis?
Mild	High	Weak	**Yes**
Cool	Normal	Weak	**Yes**
Cool	Normal	Strong	**No**
Mild	Normal	Weak	**Yes**
Mild	High	Strong	**No**



Decision Tree Learning

- Basic step: choose an attribute and, based on its values, split the data into smaller sets
  - Recursively repeat this step until we can surely decide the label

Outlook = *Sunny*

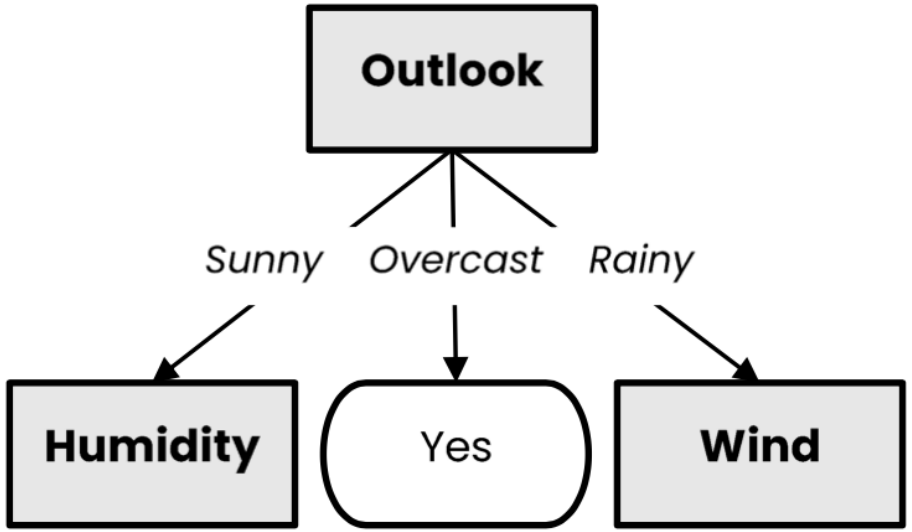
Temperature	Humidity	Wind	Play Tennis?
Hot	High	Weak	**No**
Hot	High	Strong	**No**
Mild	High	Weak	**No**
Cool	Normal	Weak	**Yes**
Mild	Normal	Strong	**Yes**

Outlook = *Overcast*

Temperature	Humidity	Wind	Play Tennis?
Hot	High	Weak	**Yes**
Cool	Normal	Strong	**Yes**
Mild	High	Strong	**Yes**
Hot	Normal	Weak	**Yes**

Outlook = *Rainy*

Temperature	Humidity	Wind	Play Tennis?
Mild	High	Weak	**Yes**
Cool	Normal	Weak	**Yes**
Cool	Normal	Strong	**No**
Mild	Normal	Weak	**Yes**
Mild	High	Strong	**No**



Decision Tree Learning

- Basic step: choose an attribute and, based on its values, split the data into smaller sets
  - Recursively repeat this step until we can surely decide the label

Outlook = *Sunny*

Humidity = High

Temperature	Wind	Play Tennis?
Hot	Weak	**No**
Hot	Strong	**No**
Mild	Weak	**No**

Humidity = Normal

Temperature	Wind	Play Tennis?
Cool	Weak	**Yes**
Mild	Strong	**Yes**

Outlook = *Overcast*

Temperature	Humidity	Wind	Play Tennis?
Hot	High	Weak	**Yes**
Cool	Normal	Strong	**Yes**
Mild	High	Strong	**Yes**
Hot	Normal	Weak	**Yes**

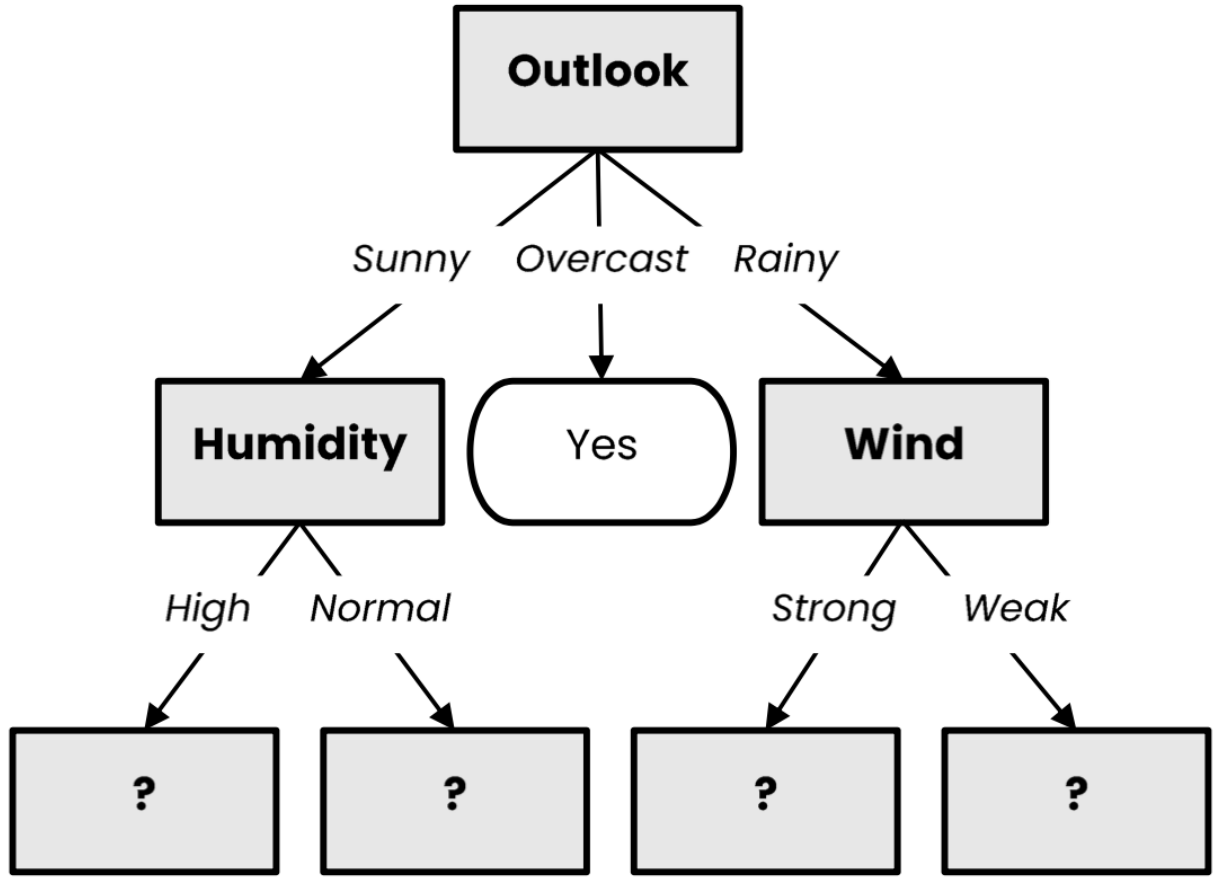
Outlook = *Rainy*

Wind = Strong

Temperature	Humidity	Play Tennis?
Cool	Normal	**No**
Mild	High	**No**

Wind = Weak

Temperature	Humidity	Play Tennis?
Mild	High	**Yes**
Cool	Normal	**Yes**
Mild	Normal	**Yes**



Decision Tree Learning

- Basic step: choose an attribute and, based on its values, split the data into smaller sets
  - Recursively repeat this step until we can surely decide the label

Outlook = *Sunny*

Humidity = High

Temperature	Wind	Play Tennis?
Hot	Weak	**No**
Hot	Strong	**No**
Mild	Weak	**No**

Humidity = Normal

Temperature	Wind	Play Tennis?
Cool	Weak	**Yes**
Mild	Strong	**Yes**

Outlook = *Overcast*

Temperature	Humidity	Wind	Play Tennis?
Hot	High	Weak	**Yes**
Cool	Normal	Strong	**Yes**
Mild	High	Strong	**Yes**
Hot	Normal	Weak	**Yes**

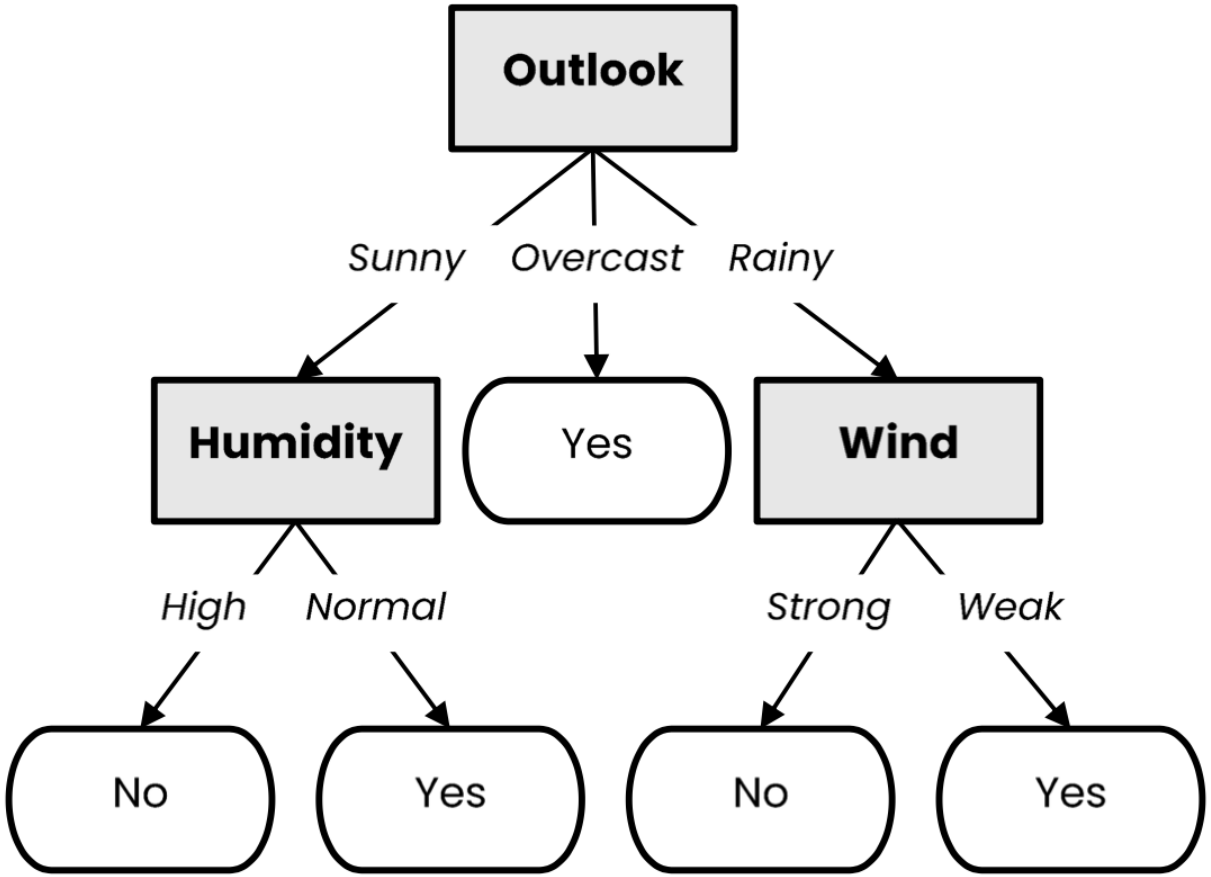
Outlook = *Rainy*

Wind = Strong

Temperature	Humidity	Play Tennis?
Cool	Normal	**No**
Mild	High	**No**

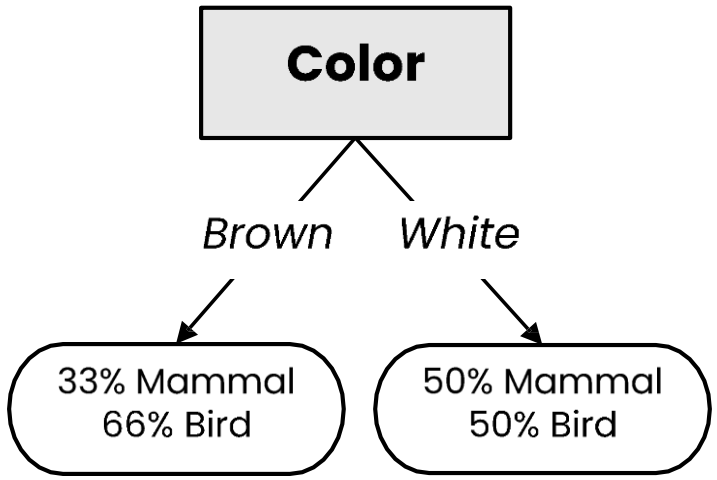
Wind = Weak

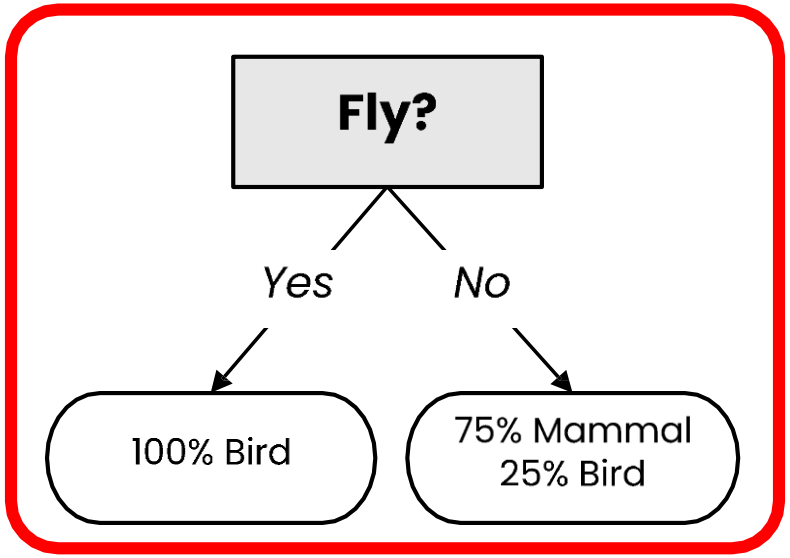
Temperature	Humidity	Play Tennis?
Mild	High	**Yes**
Cool	Normal	**Yes**
Mild	Normal	**Yes**



What is a good attribute?

Does it fly?	Color	Class
No	Brown	**Mammal**
No	White	**Mammal**
Yes	Brown	**Bird**
Yes	White	**Bird**
No	White	**Mammal**
No	Brown	**Bird**
Yes	White	**Bird**





- Which attribute provides better splitting?
- Why?
  - Because the resulting subsets are more pure
  - Knowing the value of this attribute gives us more information about the label (the entropy of the subsets is lower)

When To Stop Building the Tree?

- When all the leaf nodes are pure (leaf nodes have data that belong to one class)
- When a certain criteria is met (Such as the height of the tree exceeds certain limit, In which case the leaf nodes might not be pure but might output probability of a class instead of the class itself)
- When all the features are used.

Summary

- Decision tree: A tree-like model that illustrates series of events leading to certain decisions
- A good attribute selection leads to the better splitting of the data
- Stopping the building of tree is also important to prevent the model complexity and overfitting

Attribute selection measure

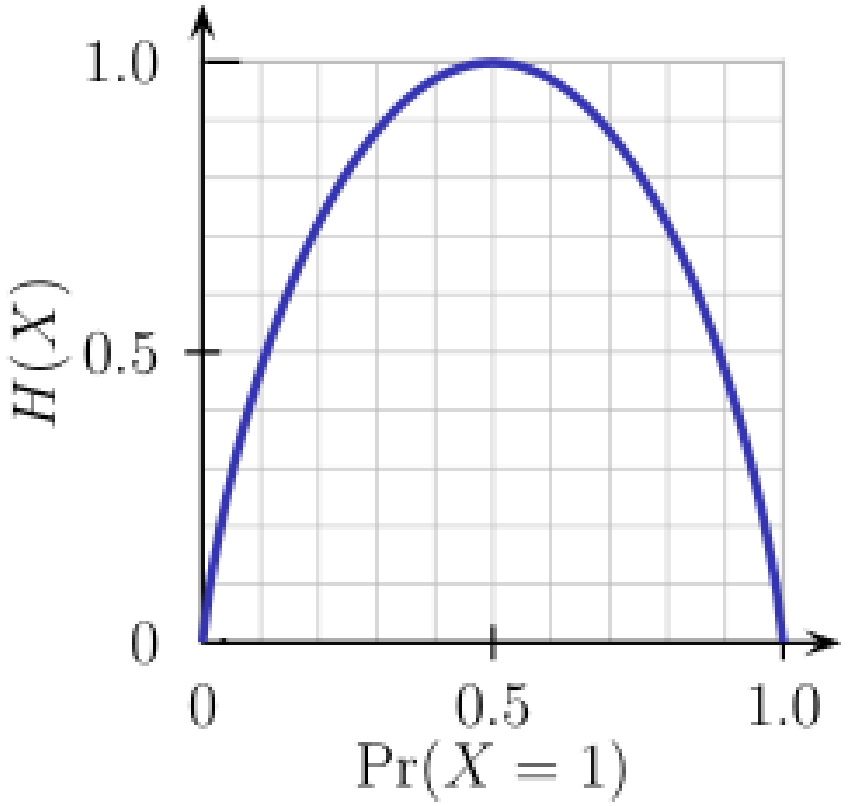
- While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems Attribute selection measure methods are used.
- There are two popular techniques for ASM, which are:
  - *Entropy and Information Gain*
  - *Gini Index*

Entropy and Information Gain

- Entropy is a measure of the randomness in the information being processed

- The higher the entropy, the harder it is to draw any conclusions from that information

- Lower entropy implies greater predictability!



E.g. Flipping a coin

Note: ID3 (Iterative Dichotomiser) follows the rule — A branch with an entropy of zero is a leaf node and A branch with entropy more than zero needs further splitting.

Entropy

- To build a decision tree, we need to calculate two types of entropy using frequency tables as follows:
  a) Entropy using the frequency table of one attribute

$$E(S) = \sum_{i=1}^{c} -pi\ log_2 p_i$$

PlayGolf	
Yes	No
9	5

Entropy (PlayGolf) = Entropy (5,9)
= Entropy (0.36, 0.64)
= -(0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)
= 0.94

Entropy

b) Entropy using the frequency table of two attributes:

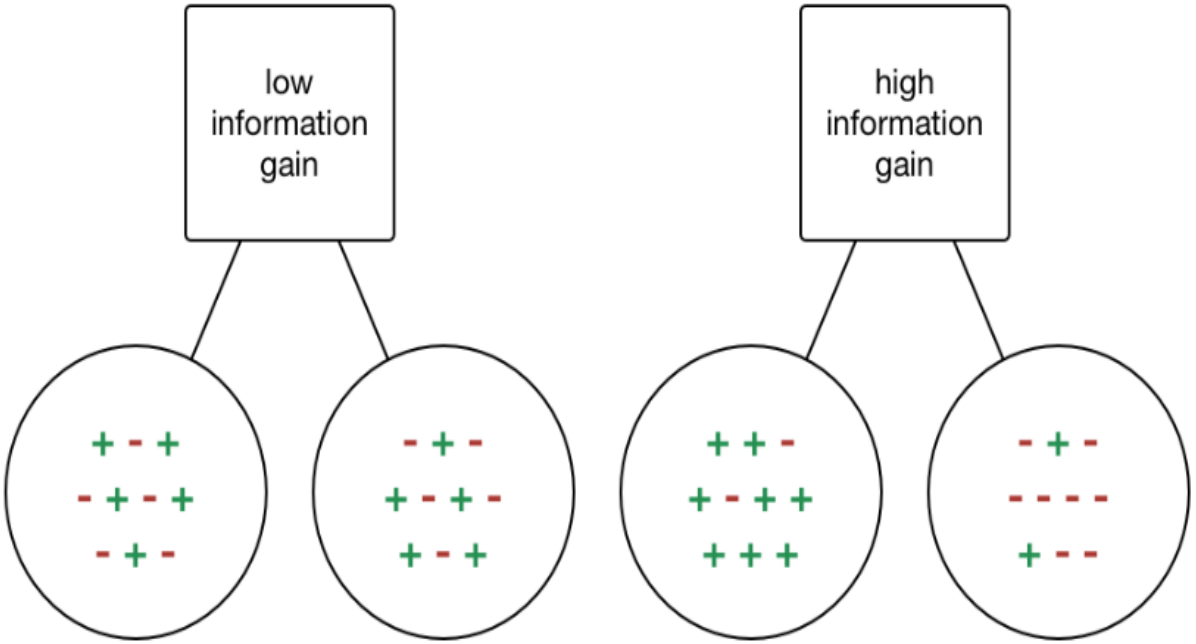
		PlayGolf		
		Yes	No	
Outlook	Sunny	2	3	5
	Overcast	4	0	4
	Rainy	3	2	5
				14

E(PlayGolf, Outlook) = $P(Sunny) * E(2,3) + P(Overcast) * E(4,0) + P(Rainy) * E(3,2)$

= (5/14)* 0.971 + (4/14) * 0.0 + (5/14) * 0.971

= 0.693

Information Gain



- Information gain or IG is a statistical property that measures how well a given attribute separates the training examples according to their target classification.
- Constructing a decision tree is all about finding an attribute that returns the highest information gain and the smallest entropy.
- Information gain is a decrease in entropy. It computes the *difference between entropy before split and average entropy after split of the dataset* based on given attribute values.

Information Gain(T, X) = Entropy(T) – Entropy(T, X)

Information Gain

- Step 1: Calculate entropy of the target.
- Step 2: The dataset is then split on the different attributes. The entropy for each branch is calculated. Then it is added proportionally, to get total entropy for the split. The resulting entropy is subtracted from the entropy before the split. The result is the Information Gain, or decrease in entropy.

		PlayGolf	
		Yes	No
Outlook	Sunny	2	3
	Overcast	4	0
	Rainy	3	2
Gain = 0.247			

		PlayGolf	
		Yes	No
Temp	Hot	2	2
	Mild	4	2
	Cool	3	1
Gain = 0.029			

		PlayGolf	
		Yes	No
Humidity	High	3	4
	Normal	6	1
Gain = 0.152			

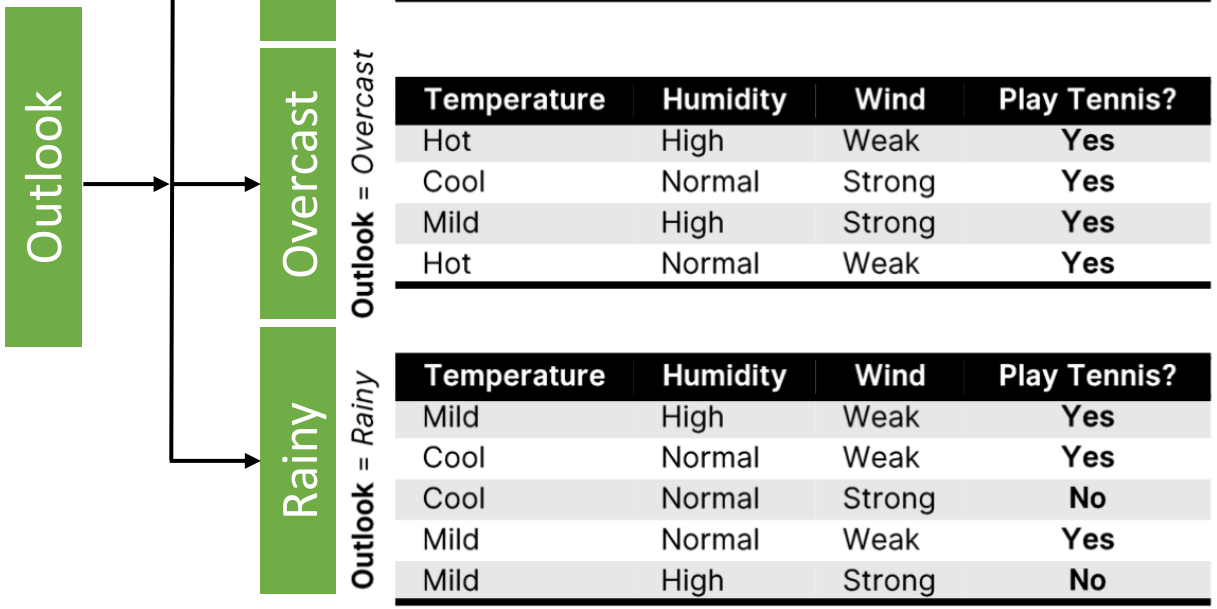
		PlayGolf	
		Yes	No
Windy	False	6	2
	True	3	3
Gain = 0.048			

$$Gain(T, X) = Entropy(T) - Entropy(T, X)$$

$$Entropy(PlayGolf) - Entropy(PlayGolf, Outlook)$$

$$= 0.940 - 0.693$$

$$= 0.247$$



Outlook = *Sunny*

Temperature	Humidity	Wind	Play Tennis?
Hot	High	Weak	**No**
Hot	High	Strong	**No**
Mild	High	Weak	**No**
Cool	Normal	Weak	**Yes**
Mild	Normal	Strong	**Yes**

Outlook = *Overcast*

Temperature	Humidity	Wind	Play Tennis?
Hot	High	Weak	**Yes**
Cool	Normal	Strong	**Yes**
Mild	High	Strong	**Yes**
Hot	Normal	Weak	**Yes**

Outlook = *Rainy*

Temperature	Humidity	Wind	Play Tennis?
Mild	High	Weak	**Yes**
Cool	Normal	Weak	**Yes**
Cool	Normal	Strong	**No**
Mild	Normal	Weak	**Yes**
Mild	High	Strong	**No**

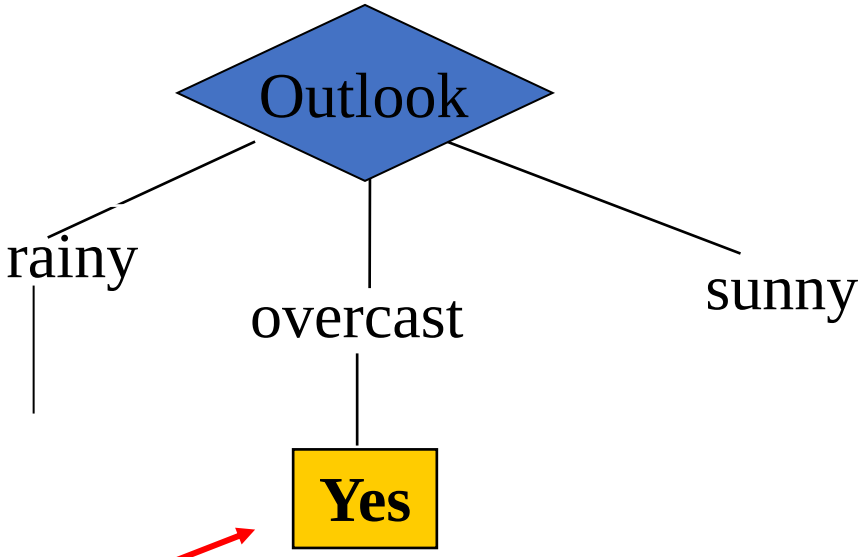
- Step 3: Choose attribute with the largest information gain as the decision node, divide the dataset by its branches and repeat the same process on every branch.

		PlayGolf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

- Step 4a: A branch with entropy of 0 is a leaf node.

Outlook = Overcast

Temperature	Humidity	Wind	Play Tennis?
Hot	High	Weak	Yes
Cool	Normal	Strong	Yes
Mild	High	Strong	Yes
Hot	Normal	Weak	Yes



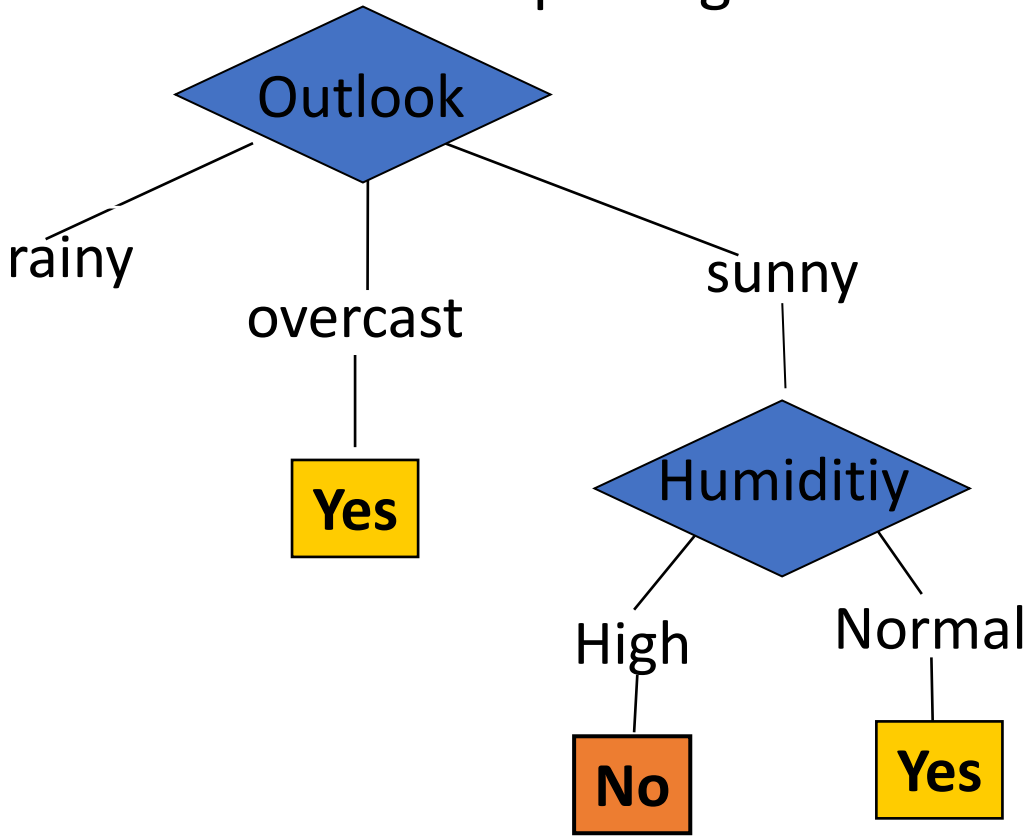
		PlayGolf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

- Step 4b: A branch with entropy more than 0 needs further splitting.

Sunny

Outlook = *Sunny*

Temperature	Humidity	Wind	Play Tennis?
Hot	High	Weak	**No**
Hot	High	Strong	**No**
Mild	High	Weak	**No**
Cool	Normal	Weak	**Yes**
Mild	Normal	Strong	**Yes**



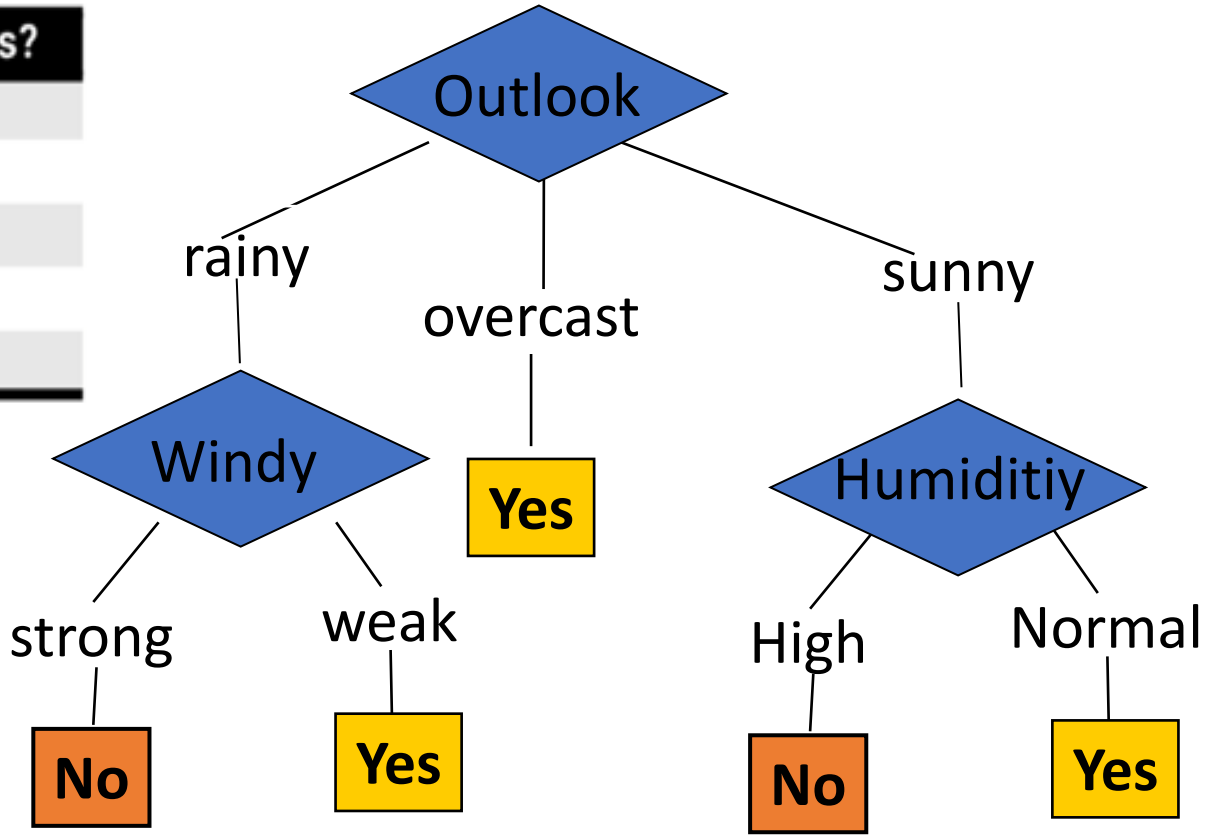
- Step 5: The ID3 algorithm is run recursively on the non-leaf branches, until all data is classified.

- Step 4b: A branch with entropy more than 0 needs further splitting.

Rainy

Outlook = *Rainy*

Temperature	Humidity	Wind	Play Tennis?
Mild	High	Weak	Yes
Cool	Normal	Weak	Yes
Cool	Normal	Strong	No
Mild	Normal	Weak	Yes
Mild	High	Strong	No



- Step 5: The ID3 algorithm is run recursively on the non-leaf branches, until all data is classified.

ID3 Algorithm (Python)

```
# ID = Iterative Dichotomiser
def ID3(X):
    node = TreeNode(X)
    if all_points_have_same_class(X):
        node.label = majority_label(X)
    else:
        a = select_attribute_with_highest_information_gain(X)
        if gain(X, a) == 0:
            node.label = majority_label(X)
        else:
            for v in values(a):
                Xv={x∈X | x[a] == v}
                node.children.append(ID3(Xv))
    return node
```

ID3 Decision Tree Learning - Explained

Day	Outlook	Temp	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

ID3 Decision Tree Learning - Explained

ID3(Examples, Target_attribute, Attributes)

- *Examples are the training examples.*
- *Target_attribute is the attribute whose value is to be predicted by the tree.*
- *Attributes is a list of other attributes that may be tested by the learned decision tree.*
- *Returns a decision tree that correctly classifies the given Examples.*

ID3 Decision Tree Learning - Explained

- Create a *Root* node for the tree
- If all *Examples are* positive, Return the single-node tree *Root,* with label → +
- If all *Examples are* negative, Return the single-node tree *Root, with* label → -
- If *Attributes* is empty, Return the single-node tree *Root,* with label = most common value of *Target_attribute in Examples*

ID3 Decision Tree Learning - Explained

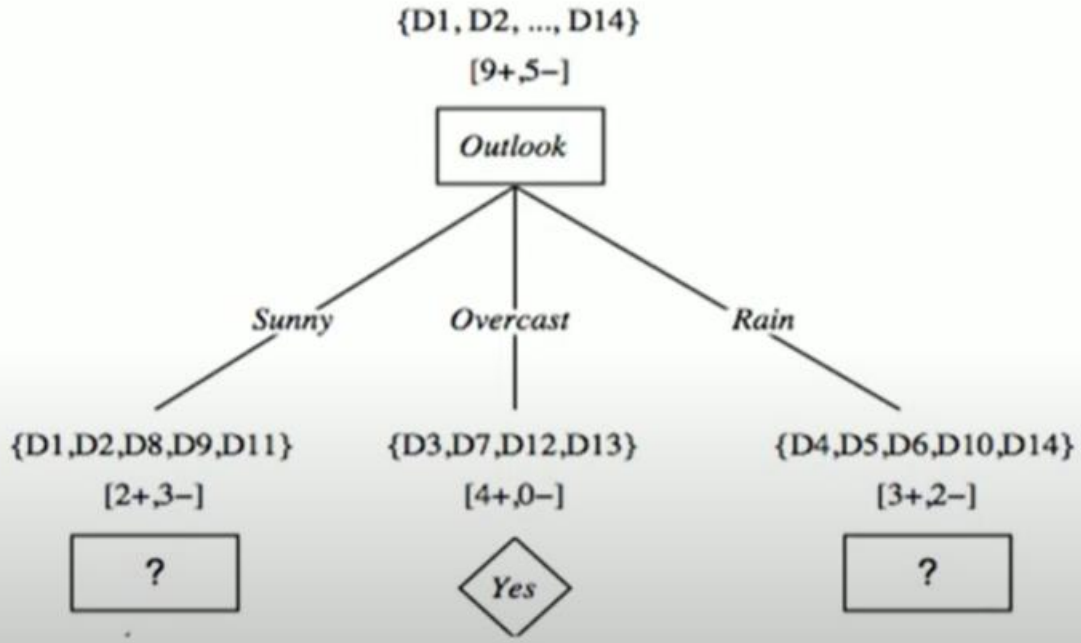
- Otherwise Begin
  - A ← the attribute from *Attributes* that best* classifies *Examples*
  - The decision attribute for *Root* ← *A*
  - For each possible value, of A,
    - Add a new tree branch below *Root,* corresponding to the test A = vi
    - Let $Examples_{vi}$ be the subset of *Examples* that have value vi for A
    - If $Examples_{vi}$ is empty
      - Then below this new branch add a leaf node with label = most common value of *Target_attribute in Examples*
    - Else
      - below this new branch add the subtree
      - *ID3*($Examples_{vi}$,*Target_attribute, Attributes - {A})*)
- End
- Return *Root*

ID3 Decision Tree Learning - Explained

Day	Outlook	Temp	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

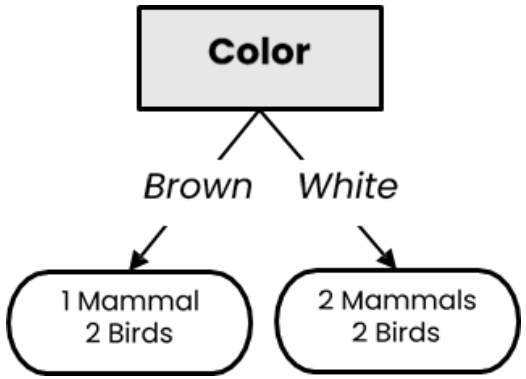
$$Gain(S, Outlook) = 0.2464,\ Gain(S, Temp) = 0.0289$$

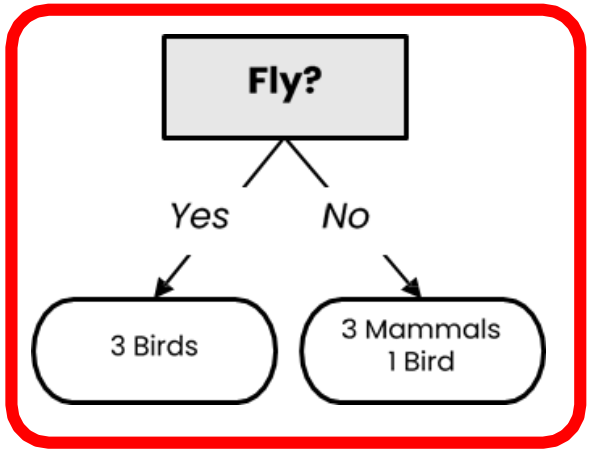
$$Gain(S, Humidity) = 0.1516,\ Gain(S, Wind) = 0.0478$$



Calculate the best attribute?

Does it fly?	Color	Class
No	Brown	Mammal
No	White	Mammal
Yes	Brown	Bird
Yes	White	Bird
No	White	Mammal
No	Brown	Bird
Yes	White	Bird





$$entropy\,(X) = -p_{\text{mammal}}\log_2 p_{\text{mammal}} - p_{\text{bird}}\log_2 p_{\text{bird}} = -\frac{3}{7}\log_2\frac{3}{7} - \frac{4}{7}\log_2\frac{4}{7} \approx 0.985$$

$$entropy\,(X_{color=brown}) = -\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} \approx 0.918 \qquad entropy\,(X_{color=white}) = 1$$

$$\boldsymbol{gain\,(X, color) = 0.985 - \frac{3}{7}\cdot 0.918 - \frac{4}{7}\cdot 1 \approx 0.020}$$

$$entropy\,(X_{fly=yes}) = 0 \qquad entropy\,(X_{fly=no}) = -\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4} \approx 0.811$$

$$\boldsymbol{gain\,(X, fly) = 0.985 - \frac{3}{7}\cdot 0 - \frac{4}{7}\cdot 0.811 \approx 0.521}$$

In practice, we compute $entropy(X)$ only once!

AMITY UNIVERSITY

Machine learning

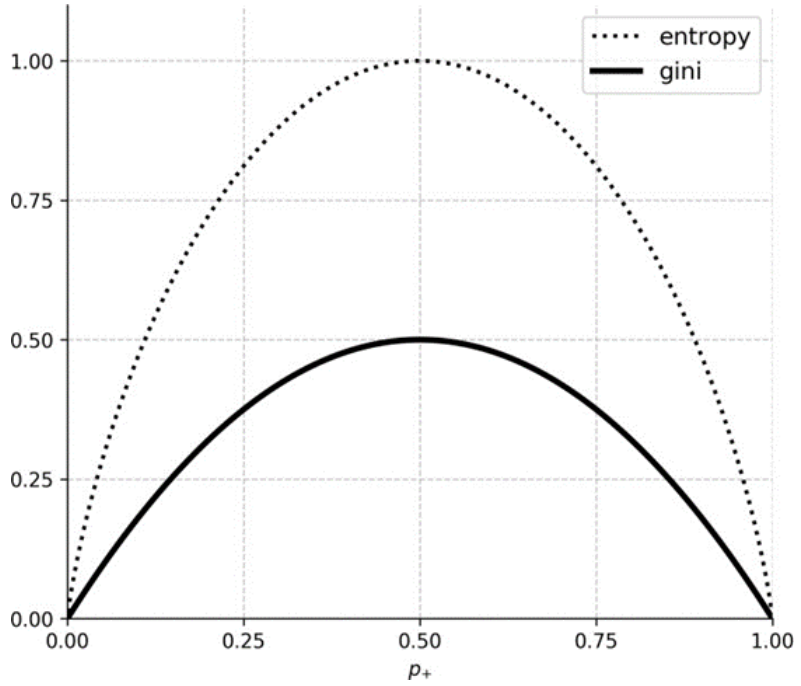
- Course Code:
- Unit 4

  Supervised Learning: Decision Trees and Ensemble Learning
- Lecture 3

  Ginni Index/Ginni Impurity

Gini Index/Gini Impurity

Error of classifying randomly picked fruit with randomly picked label



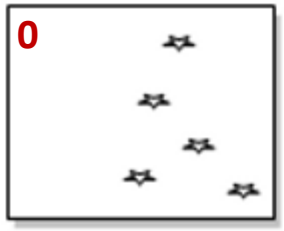
- Gini impurity measures how often a randomly chosen example would be incorrectly labeled if it was randomly labeled according to the label distribution
- Gini index as a cost function used to evaluate splits in the dataset. It favors larger partitions and easy to implement whereas information gain favors smaller partitions with distinct values.

Gini Index/Gini Impurity

- It is calculated by subtracting the sum of the squared probabilities of each class from one.
- It favors larger partitions and easy to implement whereas information gain favors smaller partitions with distinct values.
- Gini Index works with the categorical target variable “Success” or “Failure”. It performs only Binary splits.
- Calculate Gini index using: $\text{Gini Index} = 1 - \sum_j P_j^2$

  Probability of each class

- If all the elements are linked with a single class then it is called pure. It ranges from 0-1



pure

Equally distributed

Impure

Note: An attribute with a lower Gini index should be preferred.

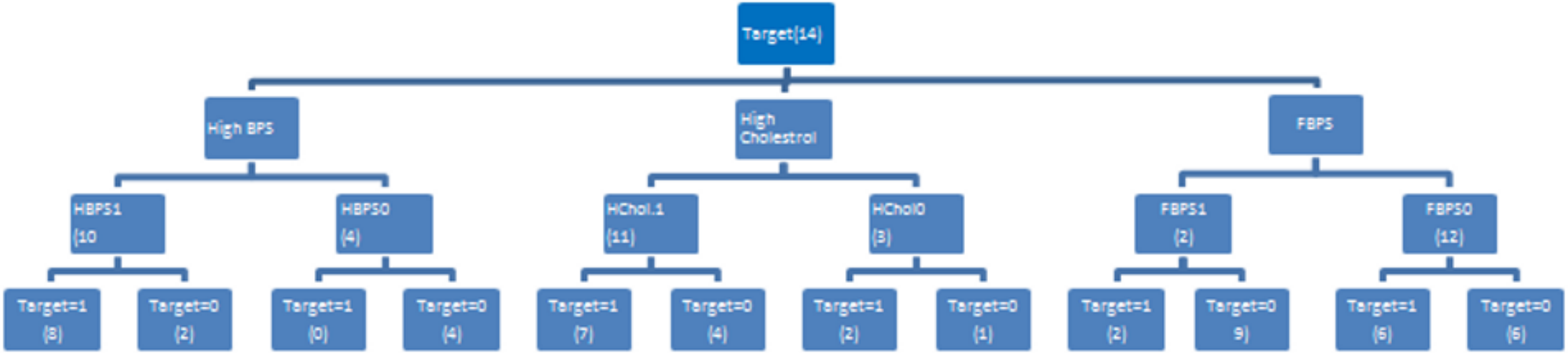
Problem: Identify the root node and calculate the gini index of each attribute?

SNO	HighBps	HighChol	Fbs	target
1	1	1	1	1
2	1	1	0	1
3	1	1	0	1
4	0	1	0	0
5	0	1	0	0
6	1	0	0	0
7	1	1	0	1
8	0	1	0	0
9	1	0	1	1
10	1	0	0	0
11	1	1	0	1
12	1	1	0	1
13	1	1	0	1
14	0	1	0	0

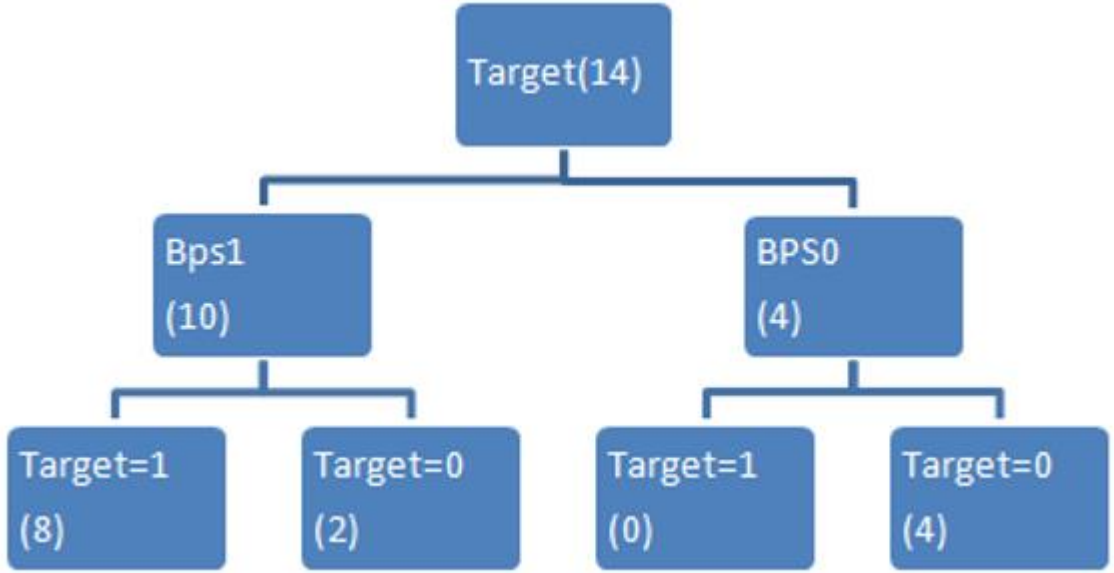
The given table represents on which factor heart disease occurs in person(target) dependent variable depending on HighBP, Highcholestrol, FBS(fasting blood sugar).

Note: The original values are converted into 1 and 0 which depict numeric classification

Decision tree for above table



- # Gini index for High Bps



If (Bps=1 and target =1) = 8/10

if(Bps=1 and target=0)=2/10

Gini index PBPS= $1 - \left((PBps1)^2 + (PBps0)^2\right)$

= 1-{{8/10)2+(2/10)2}

=0.32

Probability for parent node:

P1= 10/14

P0=4/14

Now we calculate for child node :

i) For BPS=1

for bps =1 this is the table

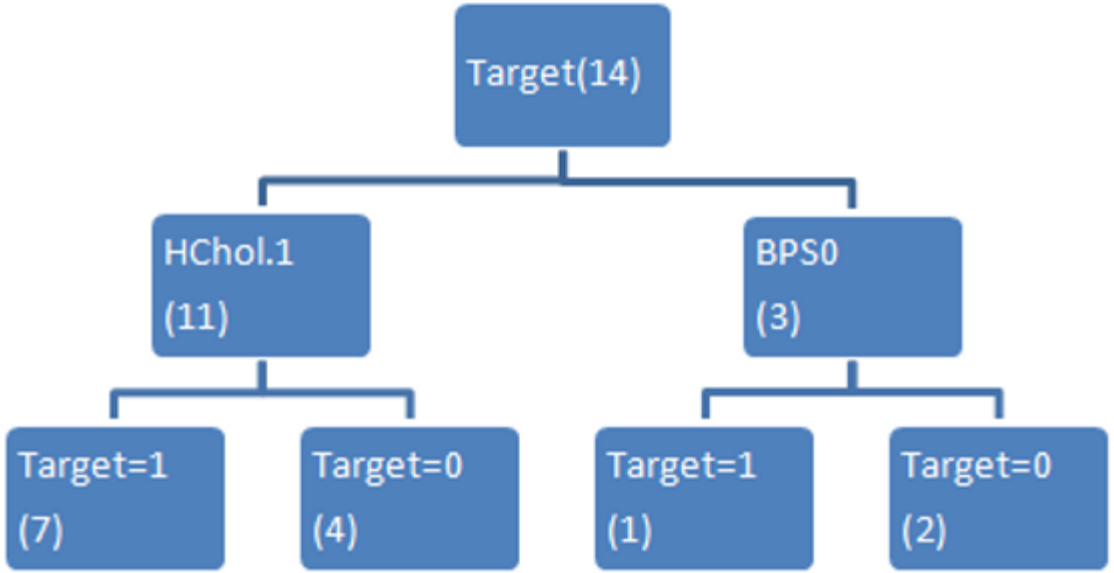
HighBps	HighChol	Fbs	Target
1	1	1	1
1	1	0	1
1	1	0	1
1	0	0	0
1	1	0	1
1	0	1	1
1	0	0	0
1	1	0	1
1	1	0	1
1	1	0	1

2) if BPS=0,

HighBps	HighChol	Fbs	Target
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0

If (BPS=0 and target=0)=4/4=1
If (BPS=0 and target=1)=0
Gini index PBPS0=1-{(1)-(0)}
= 1-1
=0
Weighted Gini index
w.g =P0*GBPS0+ P1*GBPS1
= 4/14*0 + 10/14*0.32
= 0.229

- Gini index for High Cholestrol:



If (Hchol.=1 and target=1)=7/11
If (HChol.=1 and target=0)=4/11

Gini index = 1-[(7/11)2+(4/11)2]

= 0.46

Probability of parent node
P1=11/14
P0=3/13
1) For HChol.=1

HighBps	HighChol	FBS	Target
1	1	1	1
1	1	0	1
1	1	0	1
0	1	0	0
0	1	0	0
1	1	0	1
0	1	0	0
1	1	0	1
1	1	0	1
1	1	0	1
0	1	0	0

2) If HChol.=0

HighBps	HighChol	FBS	Target
1	0	0	0
1	0	1	1
1	0	0	0

If (Hchol.=0 and target=1)=1/3

If (HChol.=0 and target=0)=2/3

Gini index= 1-[(1/3)2+(2/3)2]

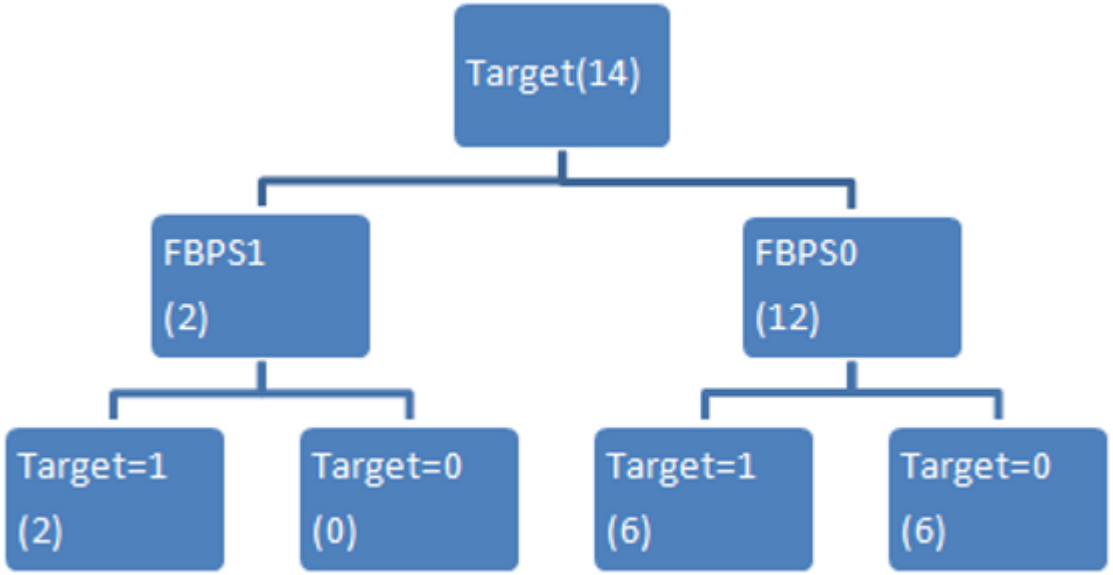
= 0.55

Weighted Gini index = P0*GHChol.0+P1*GHChol.1

= 3/14*055+11/14*0.46

= 0.47

- Gini index for FBPS:



Probability of parent node
P1=2/14
P0=12/14
1) for FBPS=1

HighBps	HighChol	Fbs	Target
1	1	1	1
1	0	1	1

If (FBps=1 and target =1)=2/2
if(FBps=1 and target=0)=0

Gini index PFBPS1=1-{(PFBPS1)2+(PFBPS0)2
= 1- [(1)2+0]
=1-1=0

2) for FBPS=0,

HighBps	HighChol	Fbs	Target
1	1	0	1
1	1	0	1
0	1	0	0
0	1	0	0
1	0	0	0
1	1	0	1
0	1	0	0
1	0	0	0
1	1	0	1
1	1	0	1
1	1	0	1
0	1	0	0

If (FBps=0 and target =1)=6/12=0.5
if(FBps=0 and target=0)=6/12=0.5

Gini index PFBPS0=1-
{(PFBPS1)2+(PFBPS0)2]
= 1-[(0.5)2+(0.5)2]

= 0.5

Weighted Gini index = P0*GFBPS0+
P1*GFBPS1
= 6/7*0.5+1/7*0

=0.42

Comparing Gini Index:

Attribute	HighBPS	Hchol.	FBPS
Gini index	0.22	0.47	0.42

As *HighBPS* is less it is the winner

Conclusion: *HighBPS* is used as the *root node* for constructing of Decision Tree and the further tree is built.

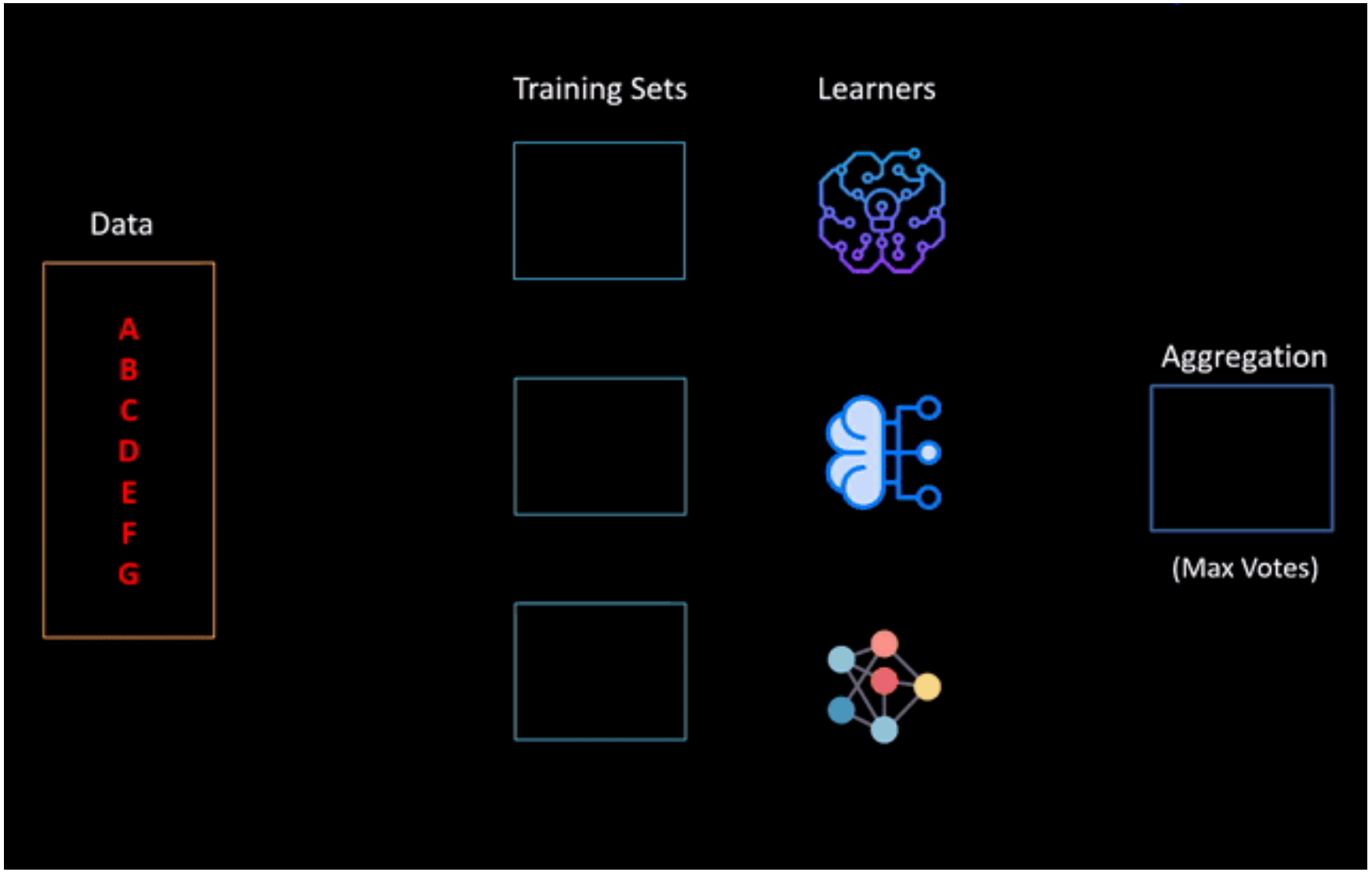
Bagging

Bagging is primarily used to solve supervised machine learning problems. It helps to reduce variance and hence avoids the possibility of overfitting. It is generally completed in two steps as follows:

- ***Bootstrapping:*** It is a random sampling method that is used to derive samples from the data using the replacement procedure. In this method, first, random data samples are fed to the primary model, and then a base learning algorithm is run on the samples to complete the learning process.

- ***Aggregation:*** This is a step that involves the process of combining the output of all base models and, based on their output, predicting an aggregate result with greater accuracy and reduced variance.

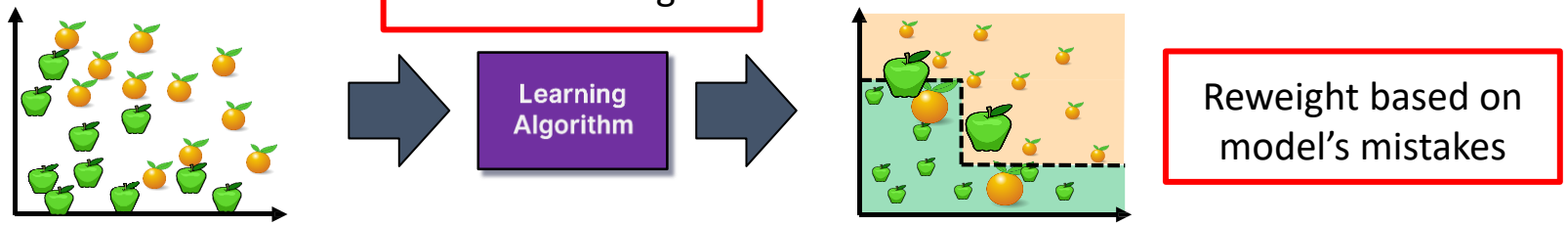
Bagging


Training Sets
Learners
Data
A
B
C
D
E
F
G
Aggregation
(Max Votes)

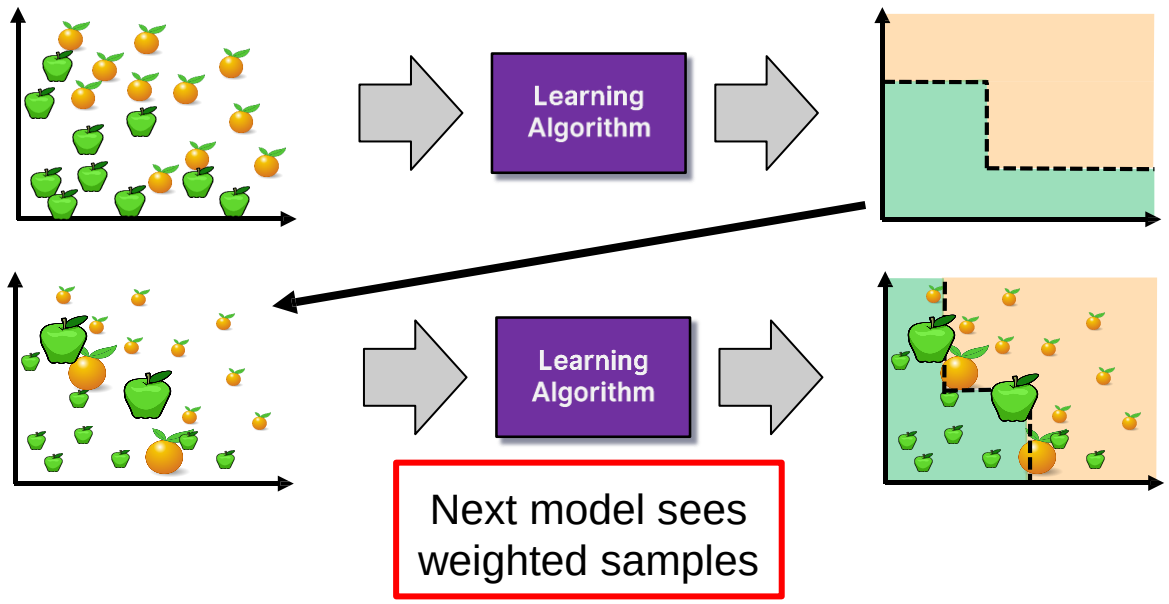
Boosting

- Boosting is an ensemble method that enables each member to learn from the preceding member's mistakes and make better predictions for the future.
- Unlike the bagging method, in boosting, all base learners (weak) are arranged in a sequential format so that they can learn from the mistakes of their preceding learner.
- Hence, in this way, all weak learners get turned into strong learners and make a better predictive model with significantly improved performance..

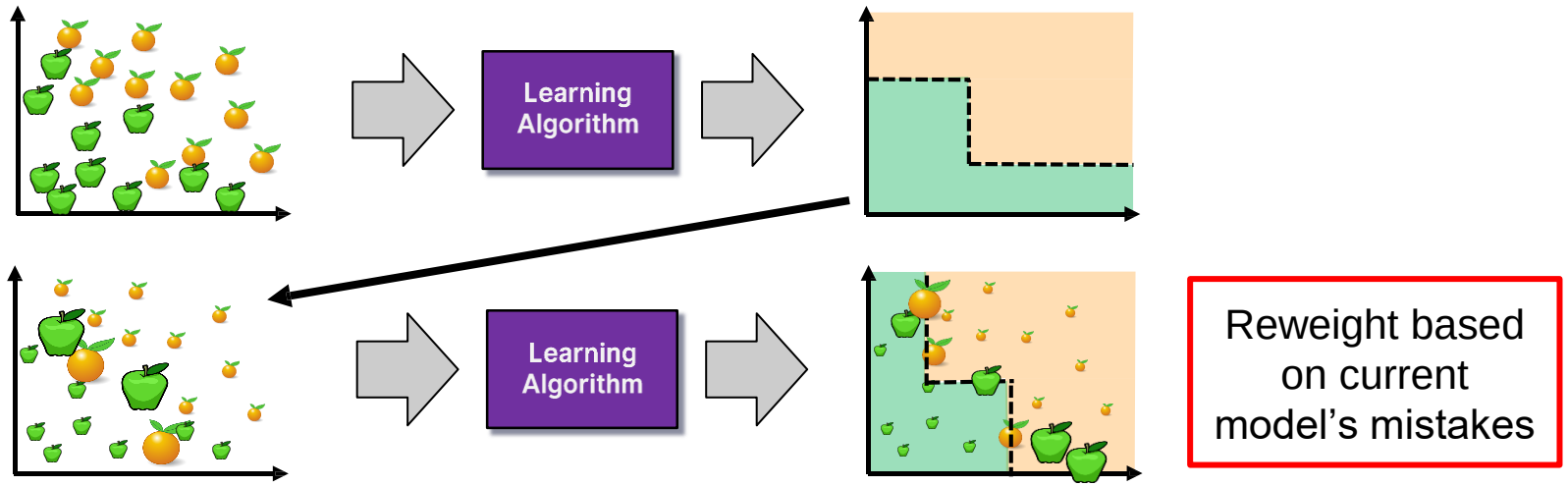
Boosting


All samples have the same weight
Learning Algorithm
Reweight based on model's mistakes

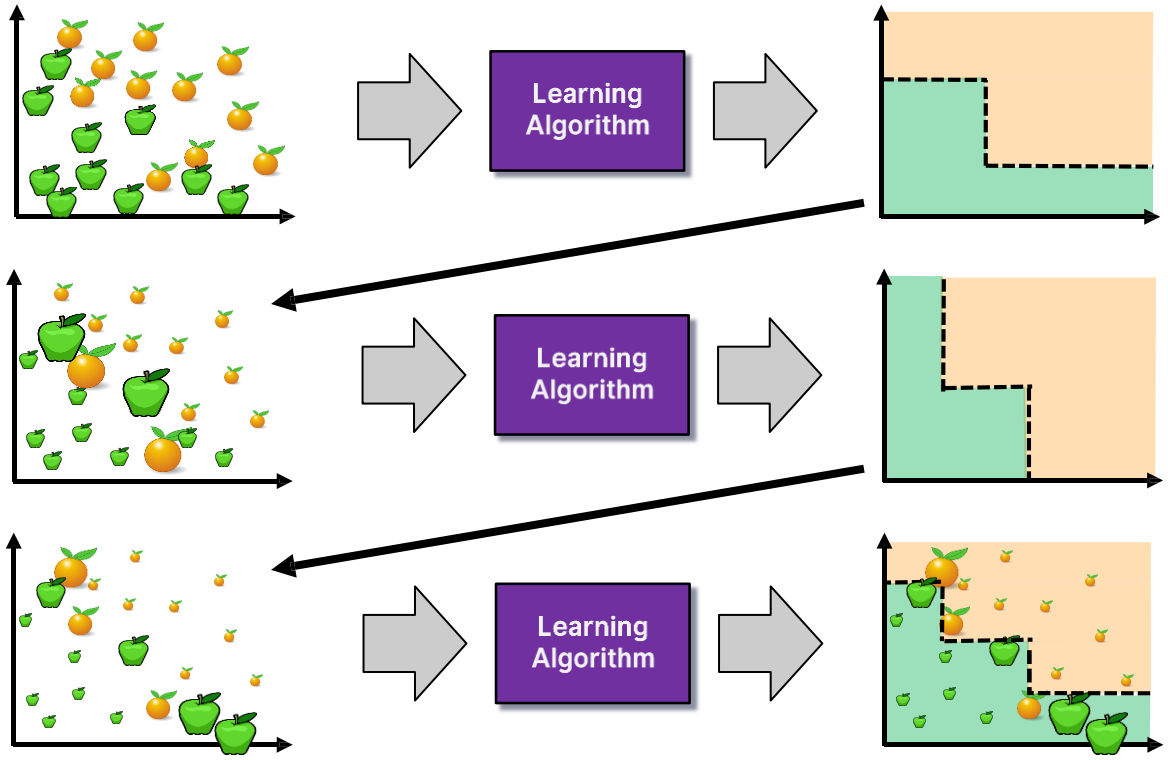
Boosting

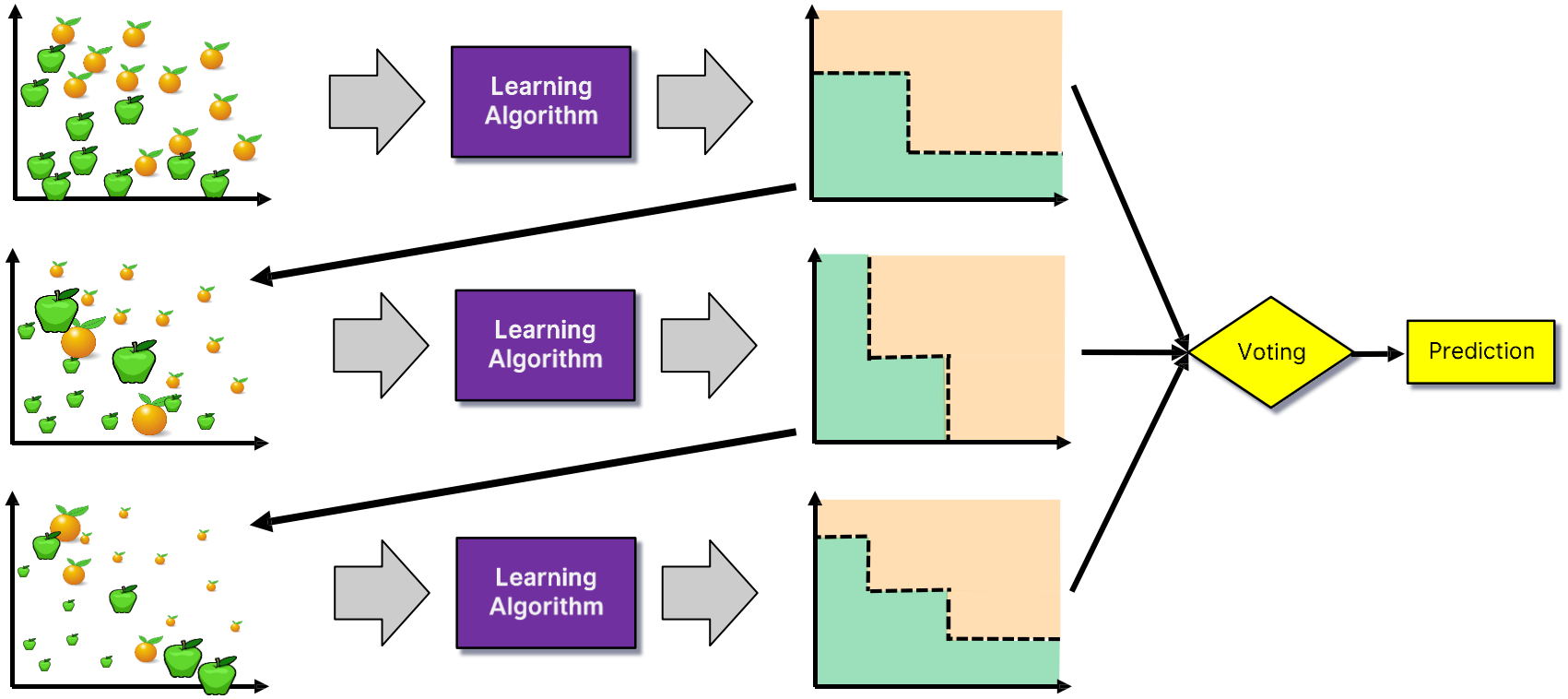

Learning Algorithm
Learning Algorithm
Next model sees weighted samples

Boosting


Learning Algorithm
Learning Algorithm
Reweight based on current model's mistakes

Boosting


Learning Algorithm
Learning Algorithm
Learning Algorithm


Learning Algorithm
Learning Algorithm
Learning Algorithm
Voting
Prediction

Random Forest

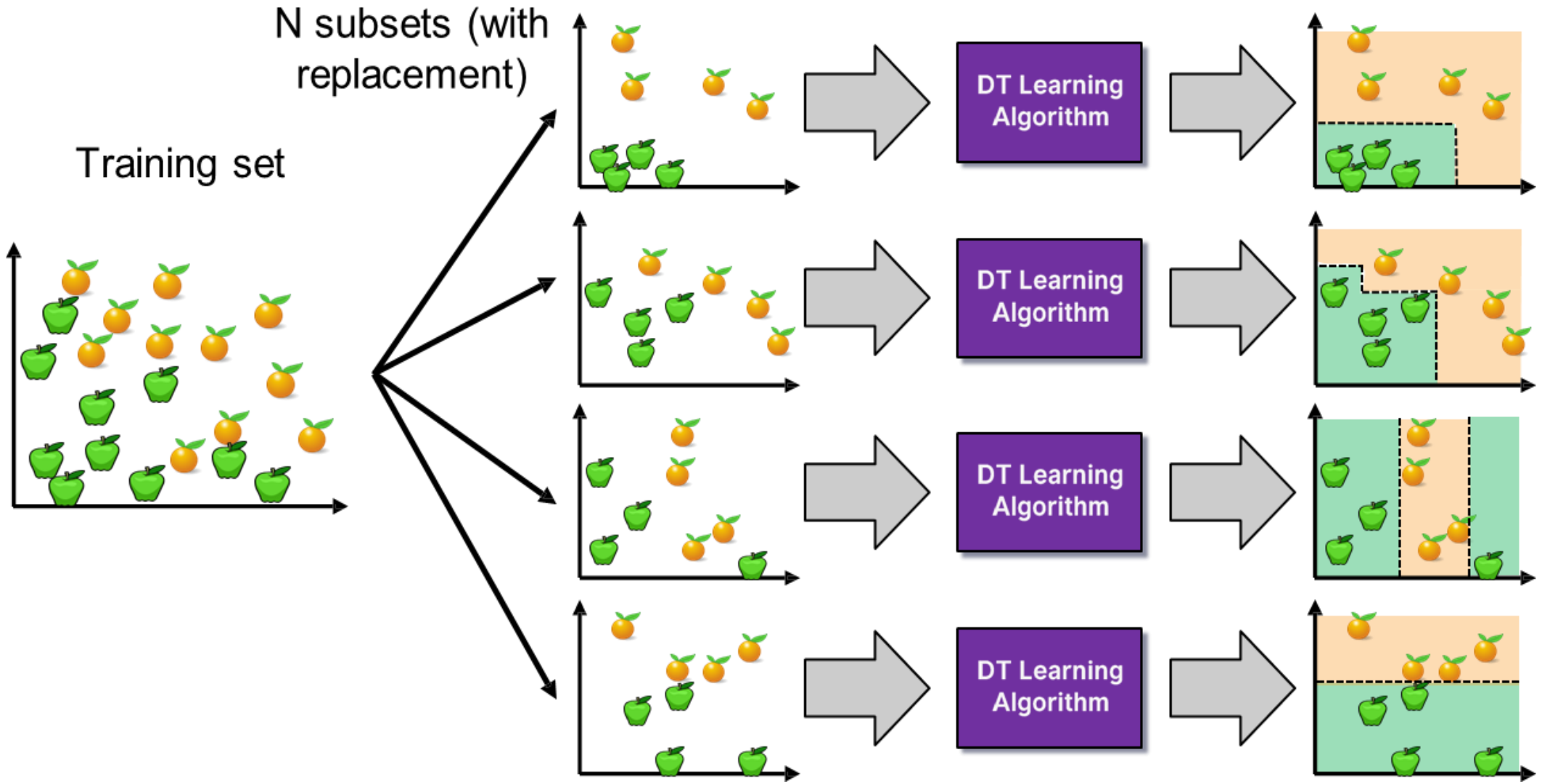
- Random Forests:
  - ➢ Instead of building a single decision tree and use it to make predictions, build many slightly different trees and combine their predictions

- We have a single data set, so how do we obtain slightly different trees?

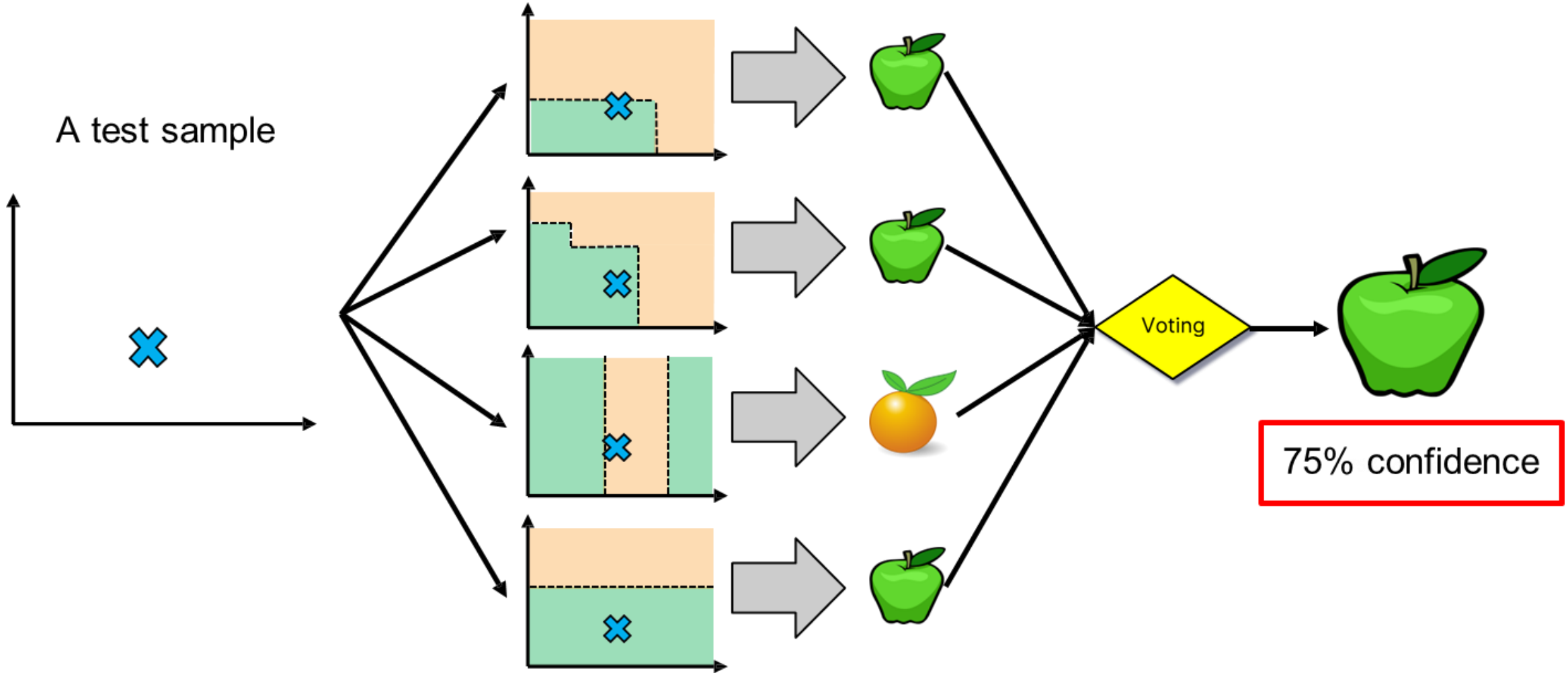
1. Bagging (**B**ootstrap **Agg**regat**ing**):
   - ➢ Take random subsets of data points from the training set to create N smaller data sets
   - ➢ Fit a decision tree on each subset

2. Random Subspace Method (also known as Feature Bagging):
   - ➢ Fit N different decision trees by constraining each one to operate on a random subset of features

Bagging at training time



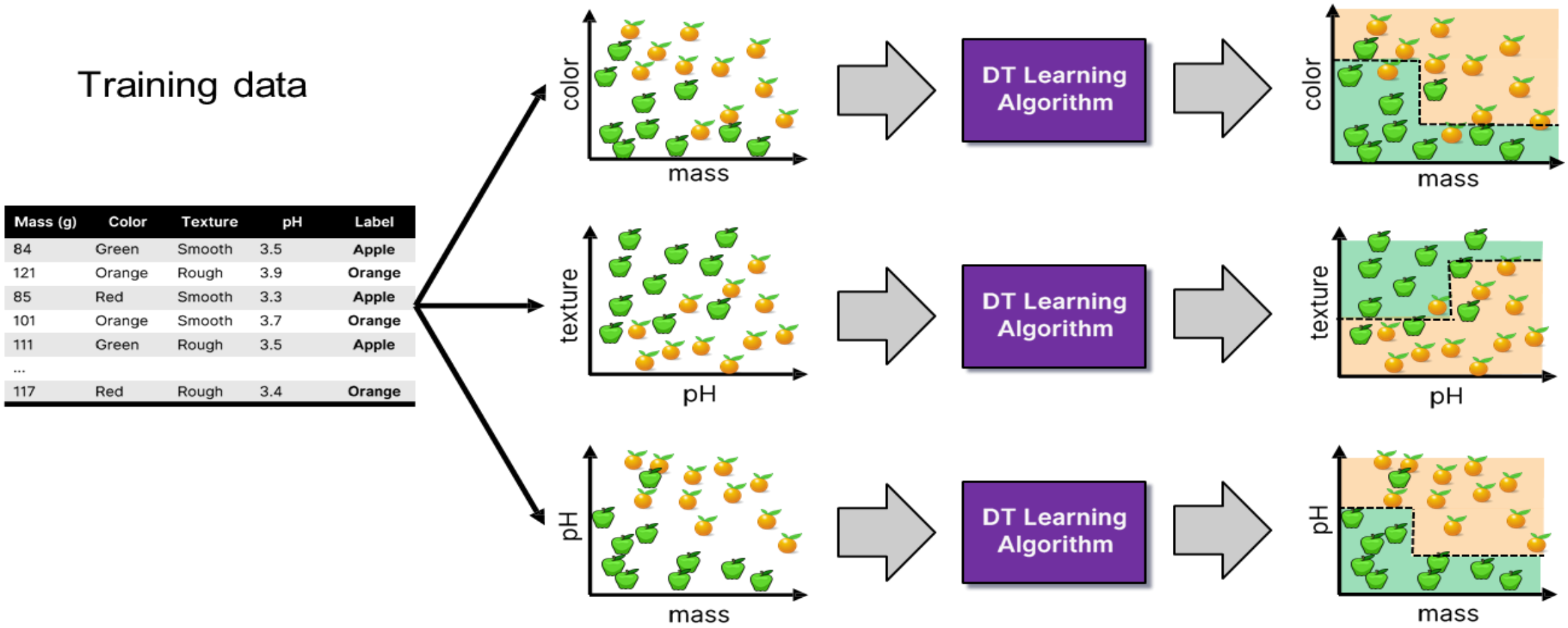
Bagging at inference time



Random Subspace Method at training time

Training data

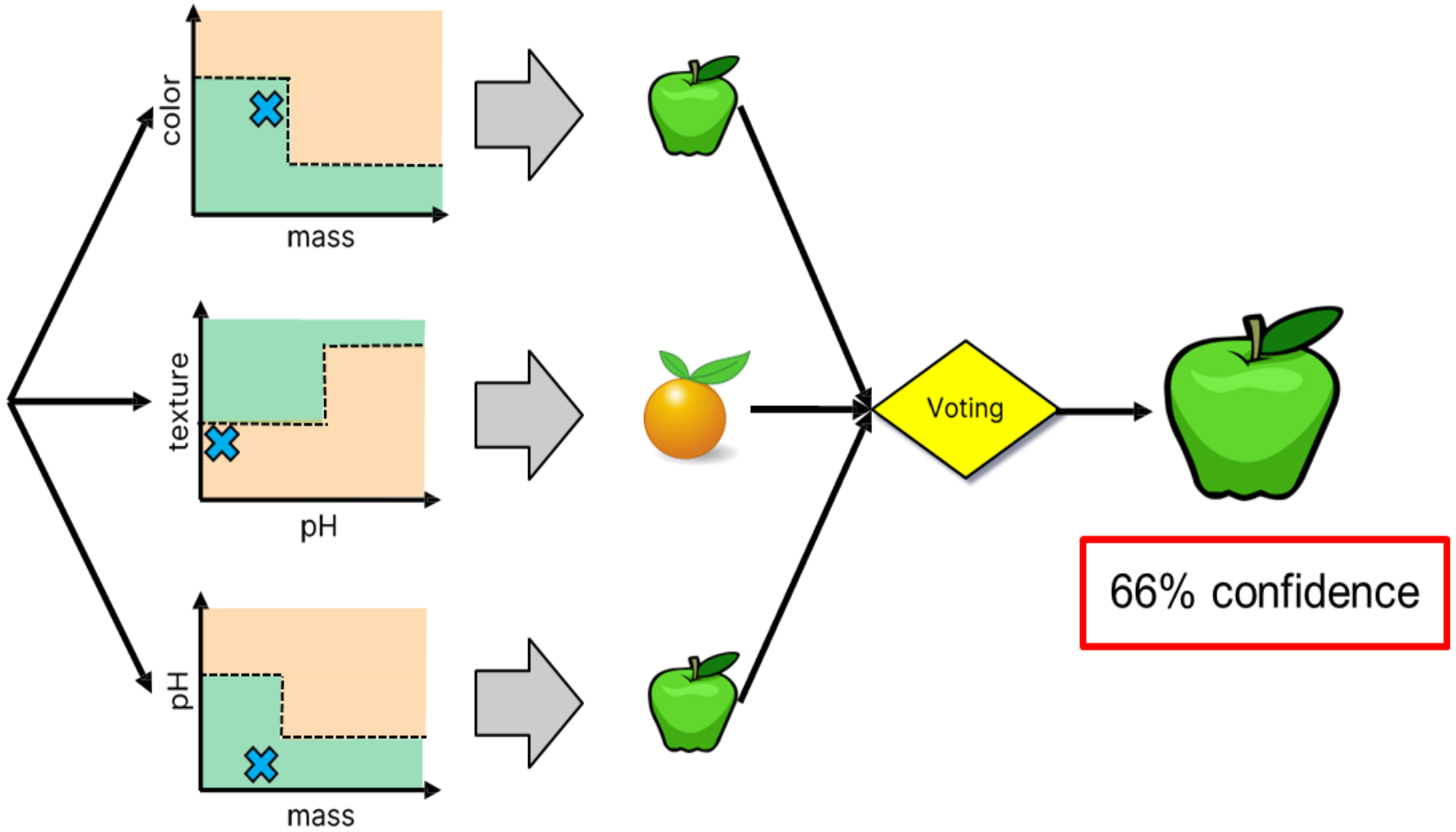
Mass (g)	Color	Texture	pH	Label
84	Green	Smooth	3.5	Apple
121	Orange	Rough	3.9	Orange
85	Red	Smooth	3.3	Apple
101	Orange	Smooth	3.7	Orange
111	Green	Rough	3.5	Apple
...				
117	Red	Rough	3.4	Orange



Random Subspace Method at inference time

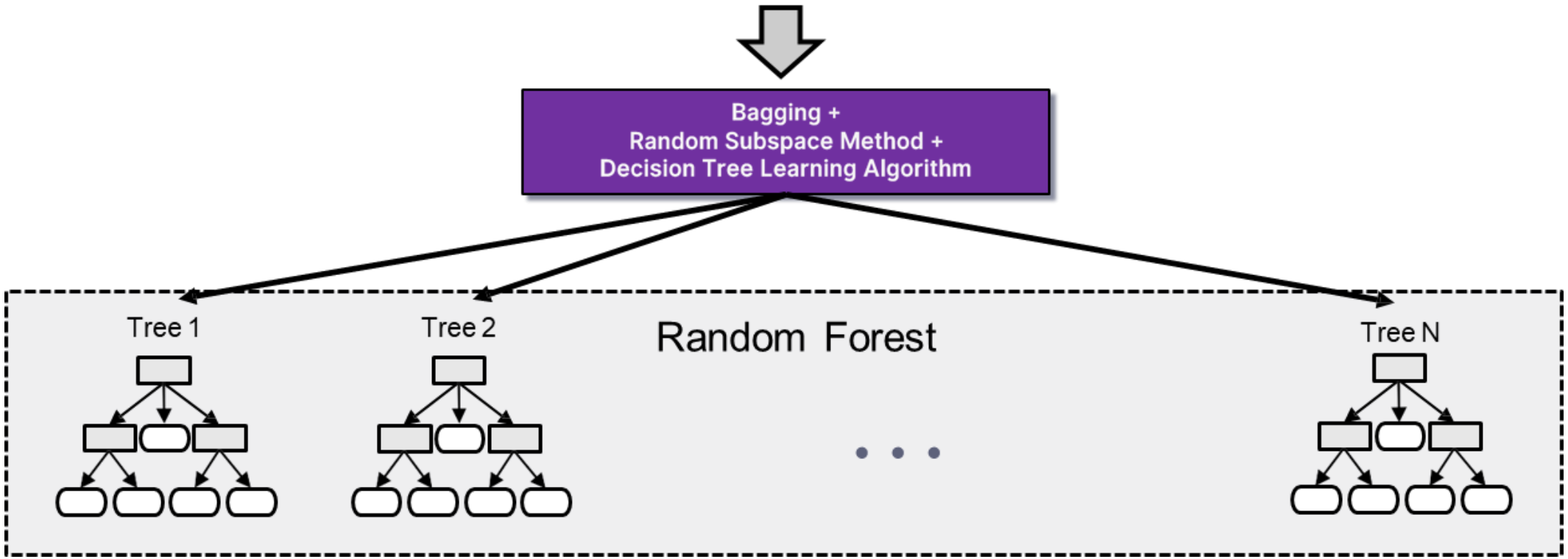
A test sample

87	Red	Smooth	3.1



Random Forest

Mass (g)	Color	Texture	pH	Label
84	Green	Smooth	3.5	**Apple**
121	Orange	Rough	3.9	**Orange**
85	Red	Smooth	3.3	**Apple**
101	Orange	Smooth	3.7	**Orange**
111	Green	Rough	3.5	**Apple**
...				
117	Red	Rough	3.4	**Orange**



Summary

Bagging	Boosting
Various training data subsets are randomly drawn with replacement from the whole training dataset.	Each new subset contains the components that were misclassified by previous models.
Bagging attempts to tackle the over-fitting issue.	Boosting tries to reduce bias.
If the classifier is unstable (high variance), then we need to apply bagging.	If the classifier is steady and straightforward (high bias), then we need to apply boosting.
Every model receives an equal weight.	Models are weighted by their performance.
Objective to decrease variance, not bias.	Objective to decrease bias, not variance.
It is the easiest way of connecting predictions that belong to the same type.	It is a way of connecting predictions that belong to the different types.
Every model is constructed independently.	New models are affected by the performance of the previously developed model.

Temporally aware distance metrics

- Temporally aware distance metrics are distance metrics that take into account the temporal relationship between the data points being compared. This is in contrast to traditional distance metrics, which do not take into account the temporal relationship between the data points.
- There are a number of different temporally aware distance metrics that have been proposed. Some of the most common ones include:
- ***Dynamic Time Warping (DTW):*** DTW is a dynamic programming algorithm that calculates the distance between two sequences by warping the time axis. This allows DTW to take into account the fact that the two sequences may not be aligned perfectly.
- ***Elastic Distance:*** Elastic Distance is a generalization of DTW that allows for more flexibility in the warping of the time axis.
- ***Time Warped Euclidean Distance (TWED):*** TWED is a distance metric that is based on DTW, but it uses the Euclidean distance instead of the dynamic programming algorithm.
- ***Shapelet Distance:*** Shapelet Distance is a distance metric that uses shapelets to represent the data points. Shapelets are small subsequences of the data points that are representative of the overall shape of the data points.

Temporally aware distance metrics can be used for a variety of tasks, such as:

- Time series classification: Temporally aware distance metrics can be used to classify time series data. For example, you could use DTW to classify time series data into different types of events, such as heart beats or speech patterns.
- Time series clustering: Temporally aware distance metrics can be used to cluster time series data. For example, you could use DTW to cluster time series data into different groups of similar time series.
- Time series similarity search: Temporally aware distance metrics can be used to search for similar time series. For example, you could use DTW to find all time series that are similar to a given time series.

Distance time wrapping (DTW)

How DTW works

DTW algorithm

- Inputs: $x_{1:N}$ and $y_{1:M}$
- Cost matrix: $\mathbf{D} \in \mathbb{R}^{(N+1)\times M+1}$
- Initialization:

  for $i = 1$ to N:

  for $j = 1$ to M:
- Calculate cost matrix:

  for $i = 1$ to N:

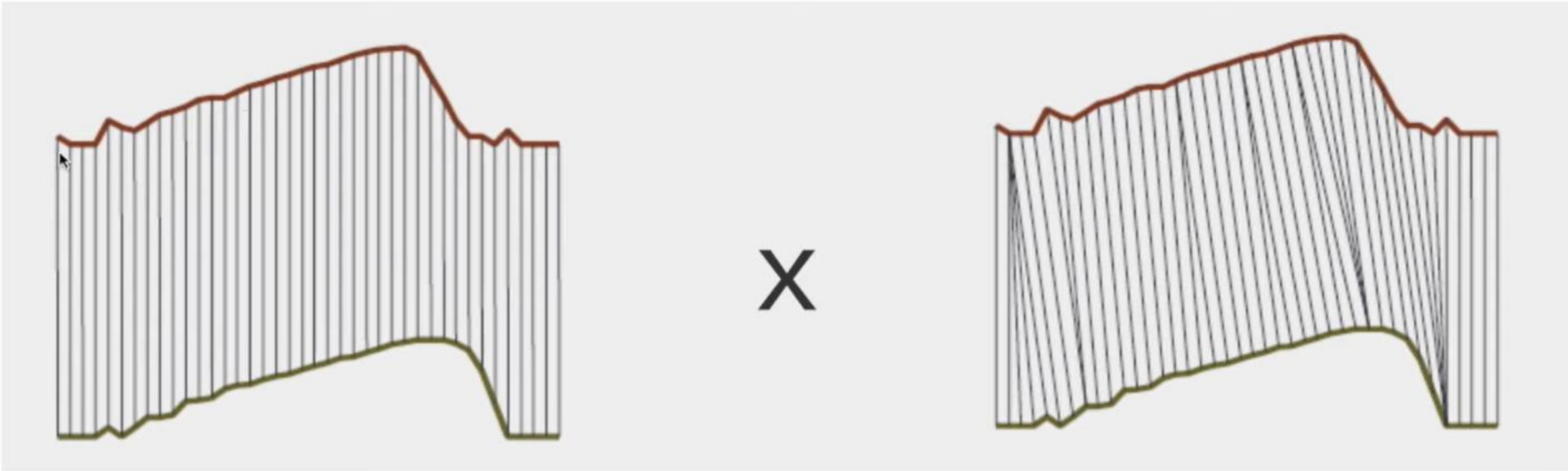
  for $j = 1$ to M:

$$D_{i,j} = d(x_i, y_j) + \min \begin{cases} D_{i-1,j-1} & \text{(match)} \\ D_{i-1,j} & \text{(insertion)} \\ D_{i,j-1} & \text{(deletion)} \end{cases}$$

- Get alignment: Trace back from $D_{N,M}$ to $D_{0,0}$

DTW

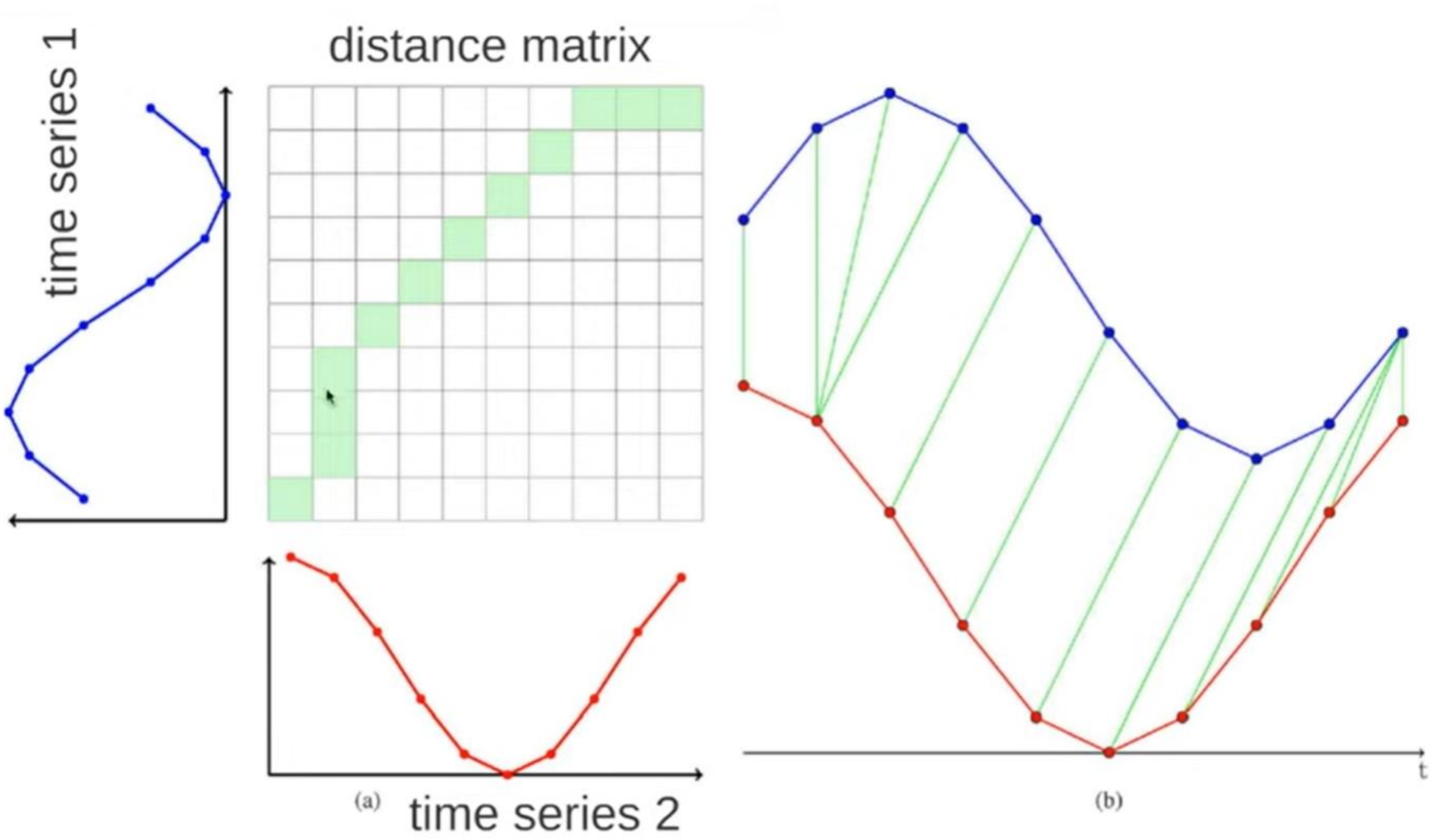
- One of the most used measure of the similarity between two time series
- Originally designed to treat automatic speech recognition
- Optimal global alignment between two time series, exploiting temporal distortions between them



Euclidean distance

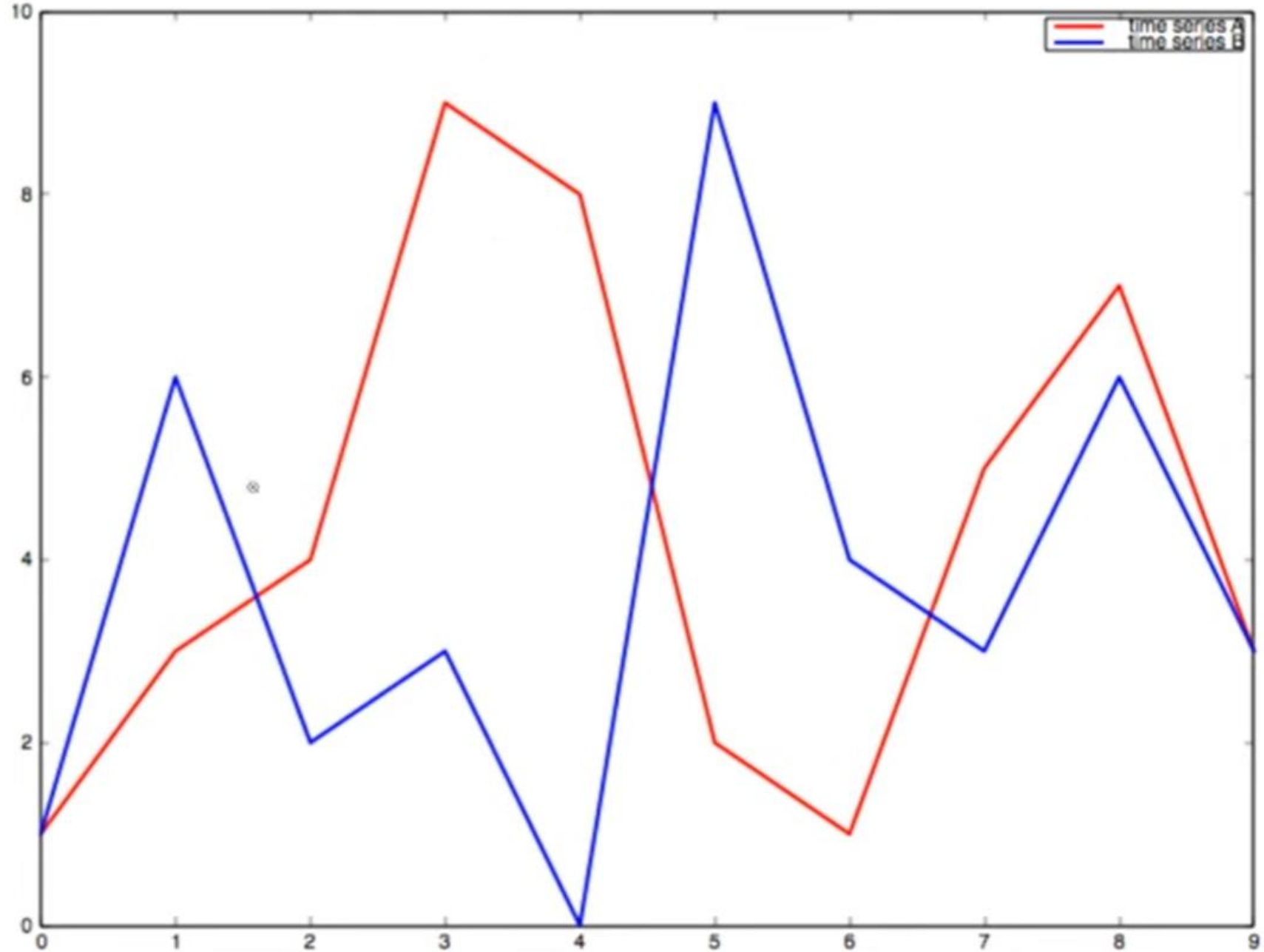
x

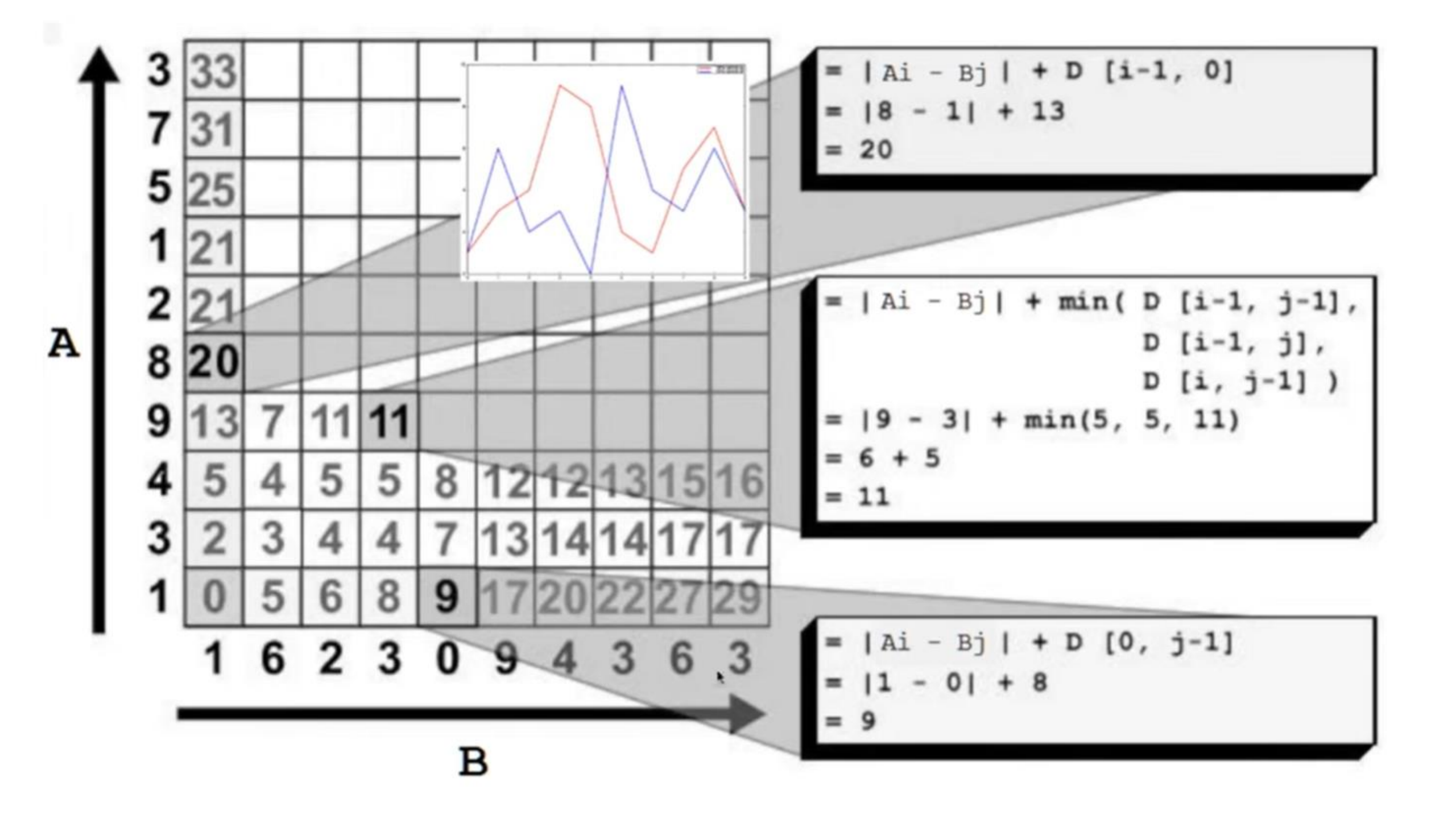
DTW

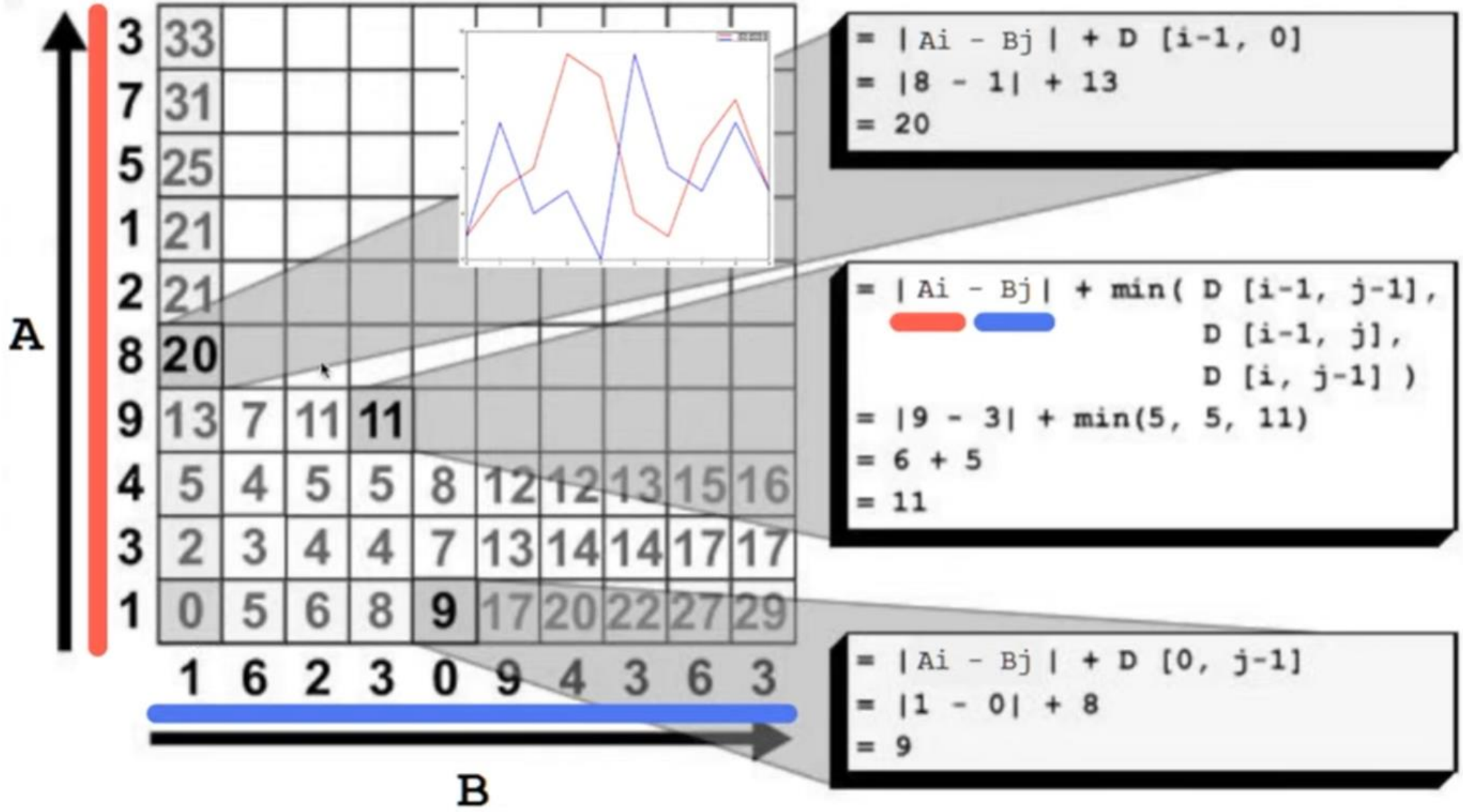

distance matrix
time series 1
time series 2
(a)
t
(b)

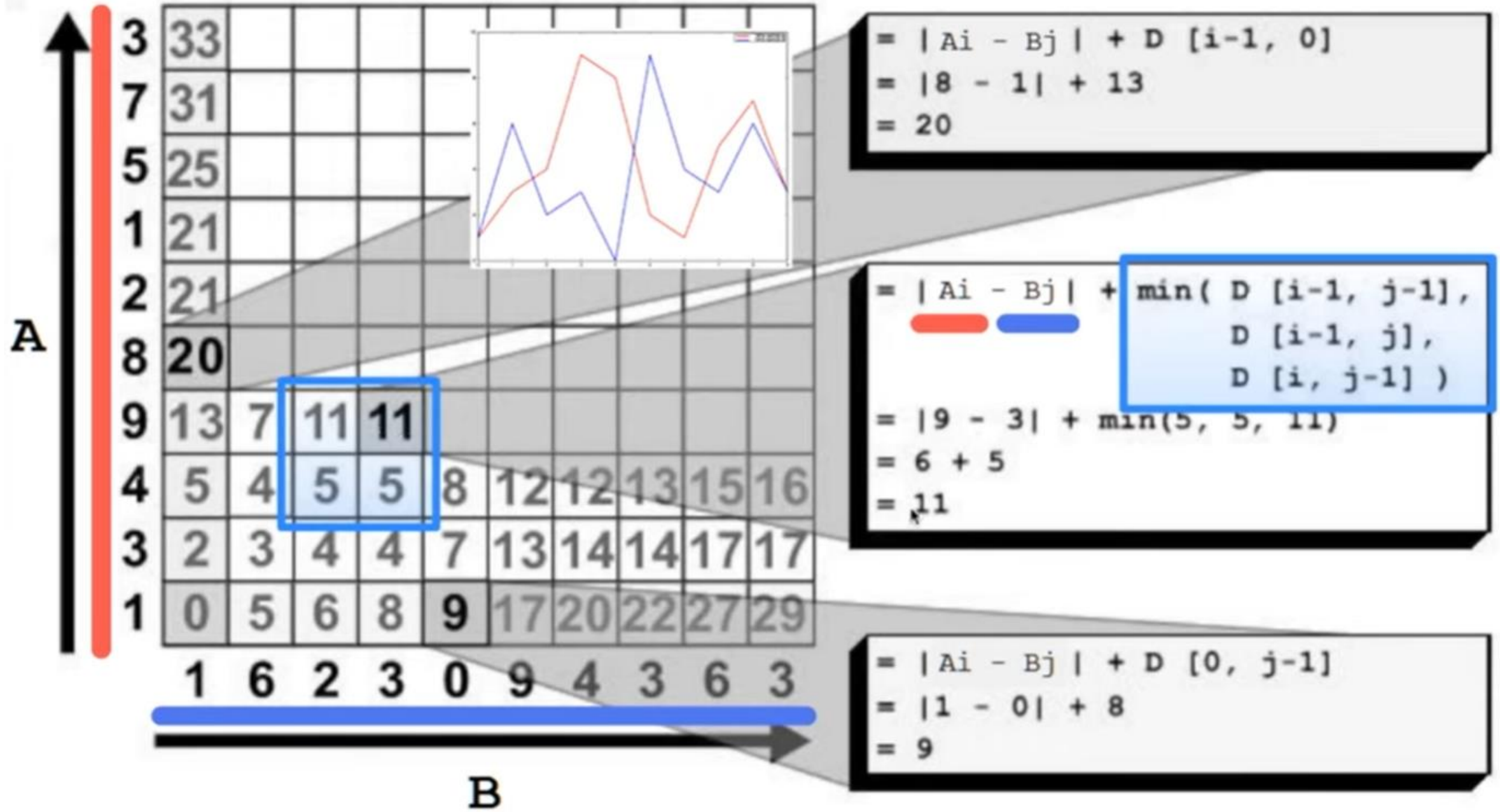
example

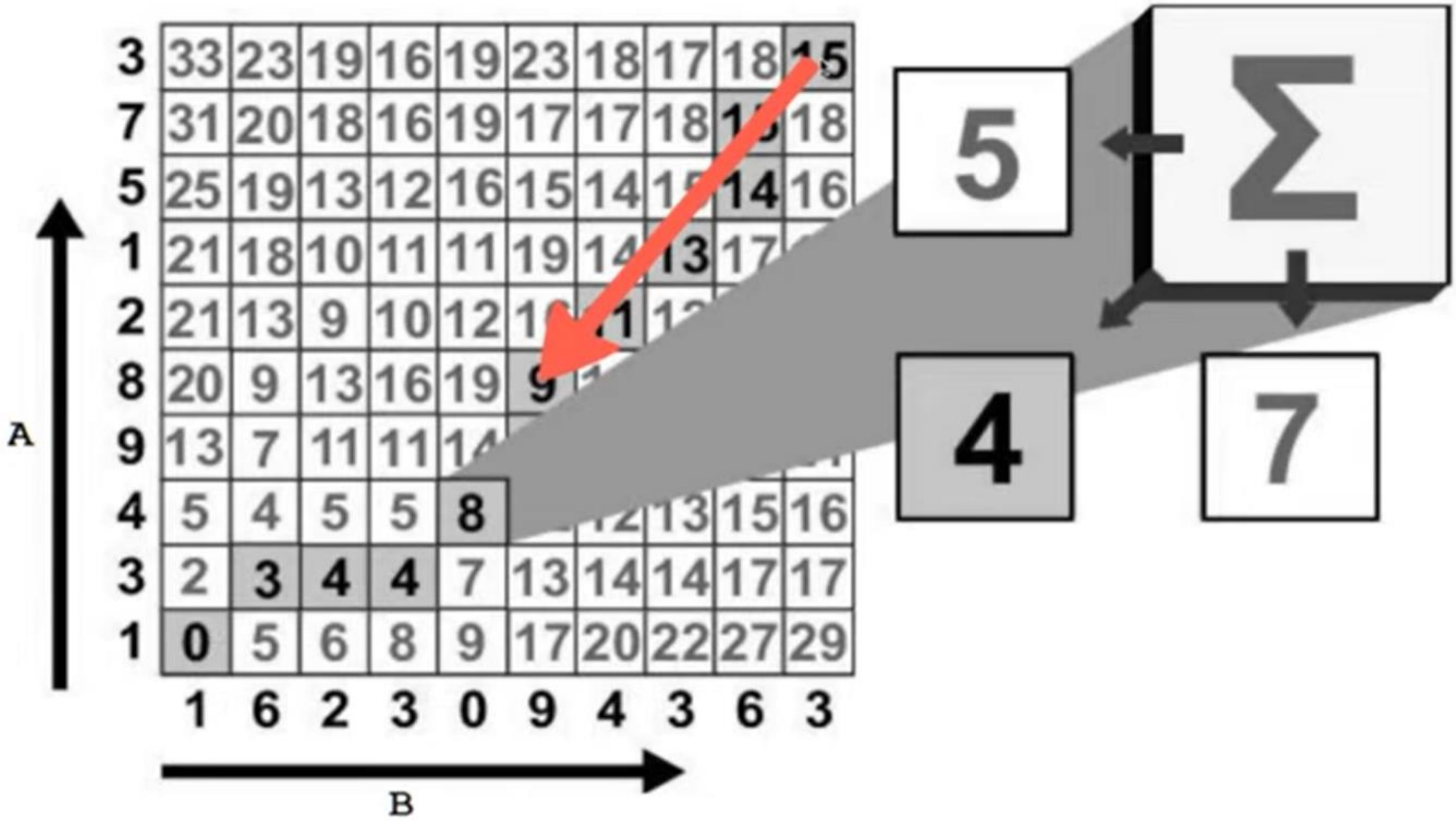
DTW between
time series A
and
time series B




A
B
3 33
7 31
5 25
1 21
2 21
8 20
9 13 7 11 11
4 5 4 5 5 8 12 12 13 15 16
3 2 3 4 4 7 13 14 14 17 17
1 0 5 6 8 9 17 20 22 27 29
1 6 2 3 0 9 4 3 6 3
= |Ai - Bj| + D [i-1, 0]
= |8 - 1| + 13
= 20
= |Ai - Bj| + min(D [i-1, j-1],
D [i-1, j],
D [i, j-1])
= |9 - 3| + min(5, 5, 11)
= 6 + 5
= 11
= |Ai - Bj| + D [0, j-1]
= |1 - 0| + 8
= 9

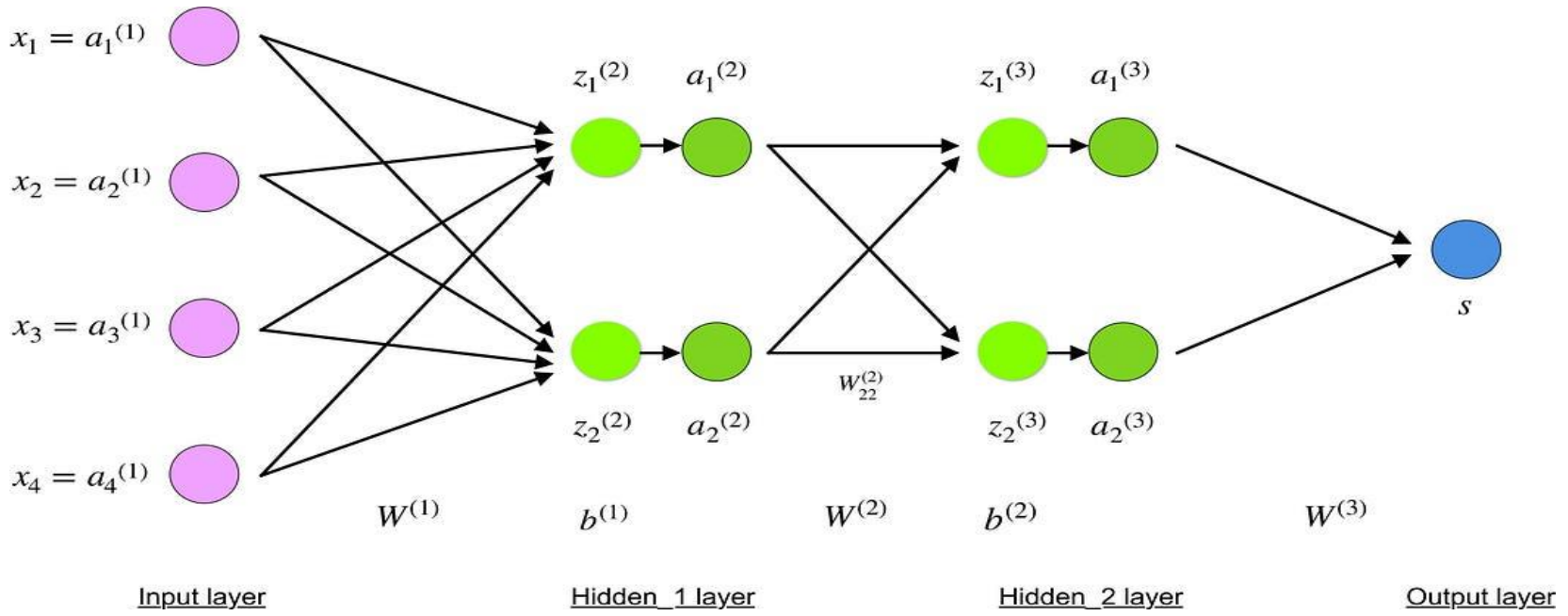

A
B
3 33
7 31
5 25
1 21
2 21
8 20
9 13 7 11 11
4 5 4 5 5 8 12 12 13 15 16
3 2 3 4 4 7 13 14 14 17 17
1 0 5 6 8 9 17 20 22 27 29
1 6 2 3 0 9 4 3 6 3
= |Ai - Bj| + D [i-1, 0]
= |8 - 1| + 13
= 20
= |Ai - Bj| + min(D [i-1, j-1],
D [i-1, j],
D [i, j-1])
= |9 - 3| + min(5, 5, 11)
= 6 + 5
= 11
= |Ai - Bj| + D [0, j-1]
= |1 - 0| + 8
= 9


A
B
3 7 5 1 2 8 9 4 3 1
1 6 2 3 0 9 4 3 6 3
33
31
25
21
21
20
13 7 11 11
5 4 5 5 8 12 12 13 15 16
2 3 4 4 7 13 14 14 17 17
0 5 6 8 9 17 20 22 27 29
= |Ai - Bj| + D [i-1, 0]
= |8 - 1| + 13
= 20
= |Ai - Bj| + min(D [i-1, j-1],
D [i-1, j],
D [i, j-1])
= |9 - 3| + min(5, 5, 11)
= 6 + 5
= 11
= |Ai - Bj| + D [0, j-1]
= |1 - 0| + 8
= 9

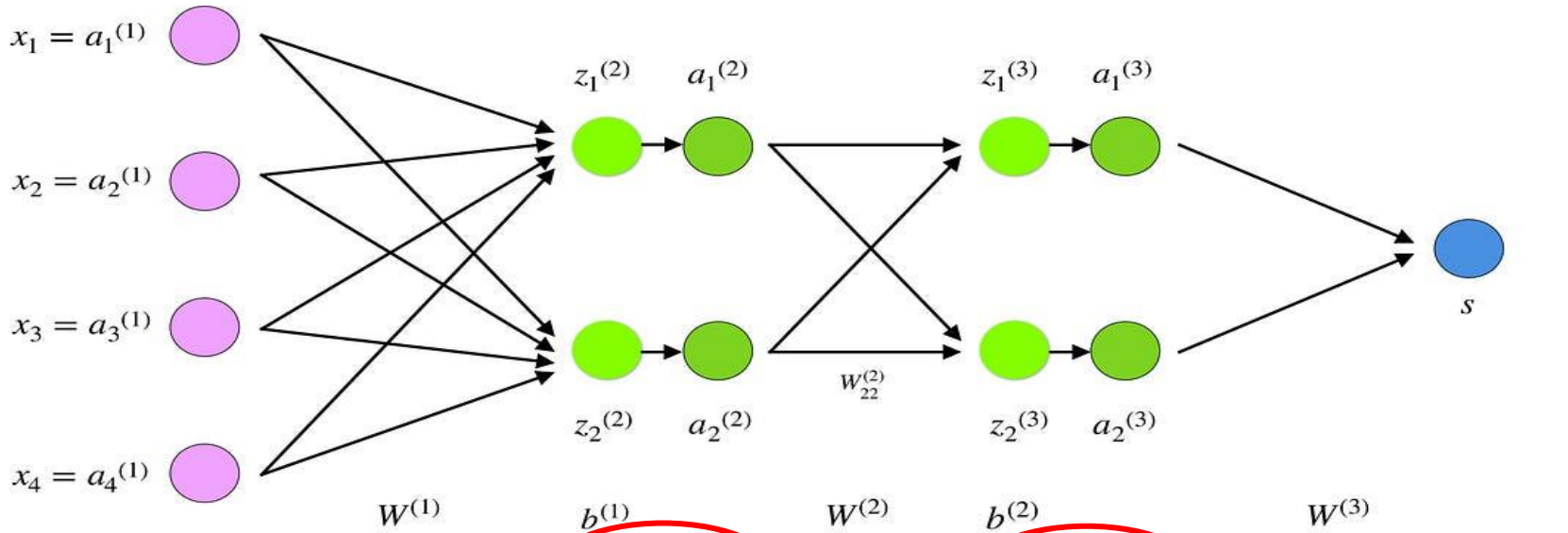

3 33 23 19 16 19 23 18 17 18 15
7 31 20 18 16 19 17 17 18 18
5 25 19 13 12 16 15 14 14 16
1 21 18 10 11 11 19 13 17
2 21 13 9 10 12
8 20 9 13 16 19 9
9 13 7 11 11
4 5 4 5 5 8 13 15 16
3 2 3 4 4 7 13 14 14 17 17
1 0 5 6 8 9 17 20 22 27 29
1 6 2 3 0 9 4 3 6 3
A
B
5
Σ
4
7

Training Neural Networks

Eg. 4-layer neural network



Eg. 4-layer neural network

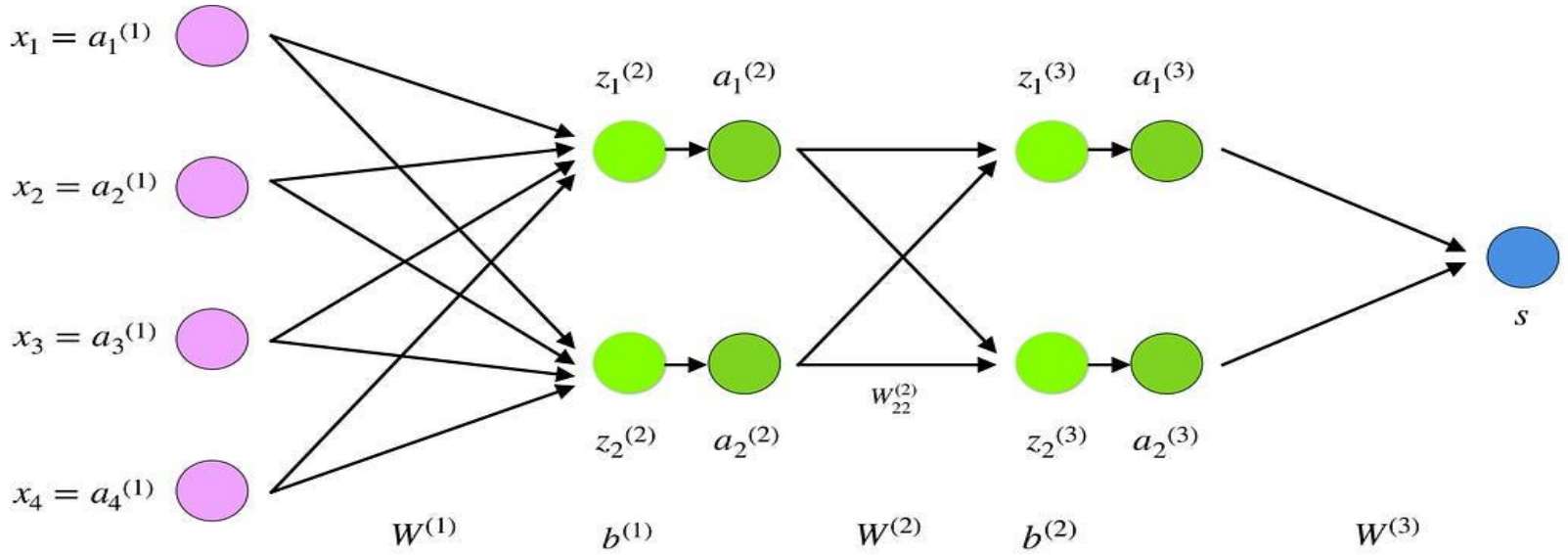


$$z^{(2)} = W^{(1)}x + b^{(1)} \qquad z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$

$$a^{(2)} = f(z^{(2)}) \qquad a^{(3)} = f(z^{(3)})$$

- W^2, W^3, b^2, b^3 - weights and biases in layer 2, 3
- Activations a^2, a^3 computed using activation function f. Typically, non-linear (e.g. sigmoid, ReLU, tanh)

Eg. 4-layer neural network

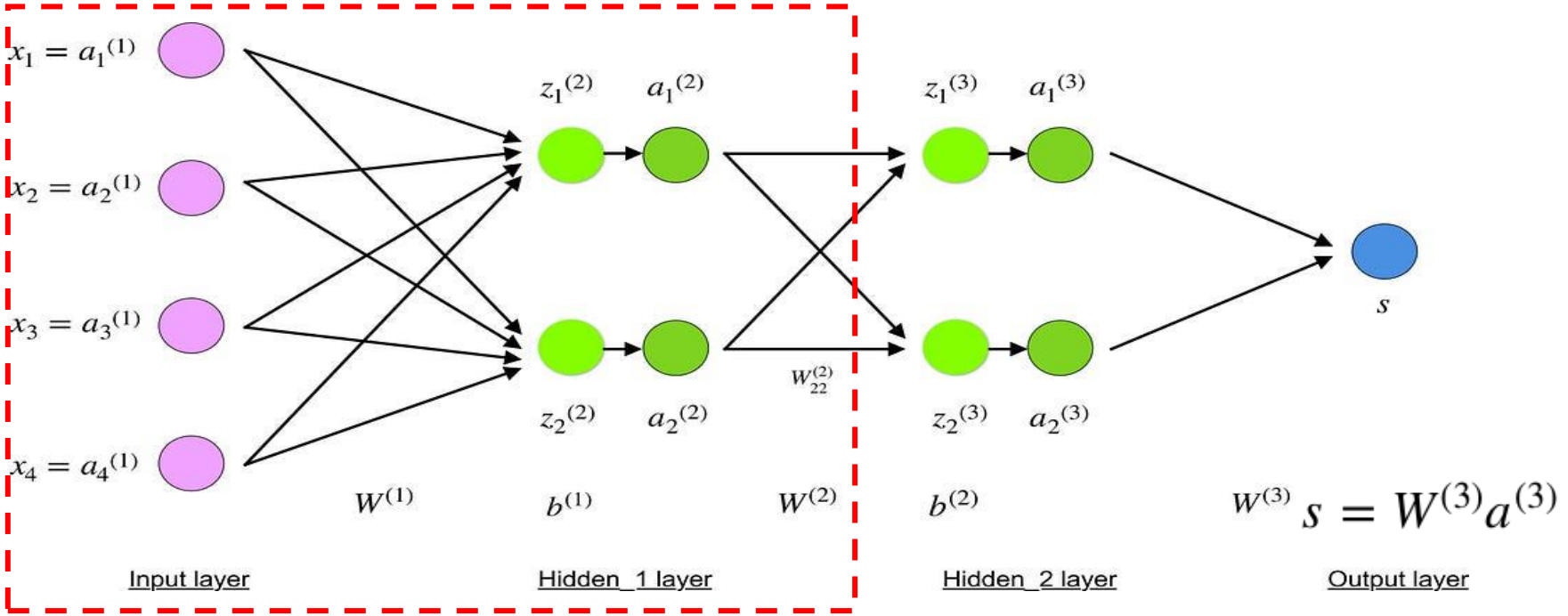


Input layer | Hidden_1 layer | Hidden_2 layer | Output layer

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \quad W^{(1)} = \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} & W_{14}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} & W_{24}^{(1)} \end{bmatrix} \quad b^{(1)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix} \quad s = W^{(3)} a^{(3)}$$

- W^1 is a weight matrix of shape (n, m): n, m - no. of output & input neurons (neurons of next & previous layer)
- x input vector of shape $(m, 1)$, m : no. of input neurons
- b^1 bias vector of shape $(n, 1)$,n: no. of neurons in current layer

Eg. 4-layer neural network



$$z^{(2)} = \begin{bmatrix} W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + W_{14}^{(1)}x_4 \\ W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + W_{24}^{(1)}x_4 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \end{bmatrix}$$

Forward Propagation and evaluation

Overview of forward propagation equations colored by layer

$x = a^{(1)}$ *Input layer*

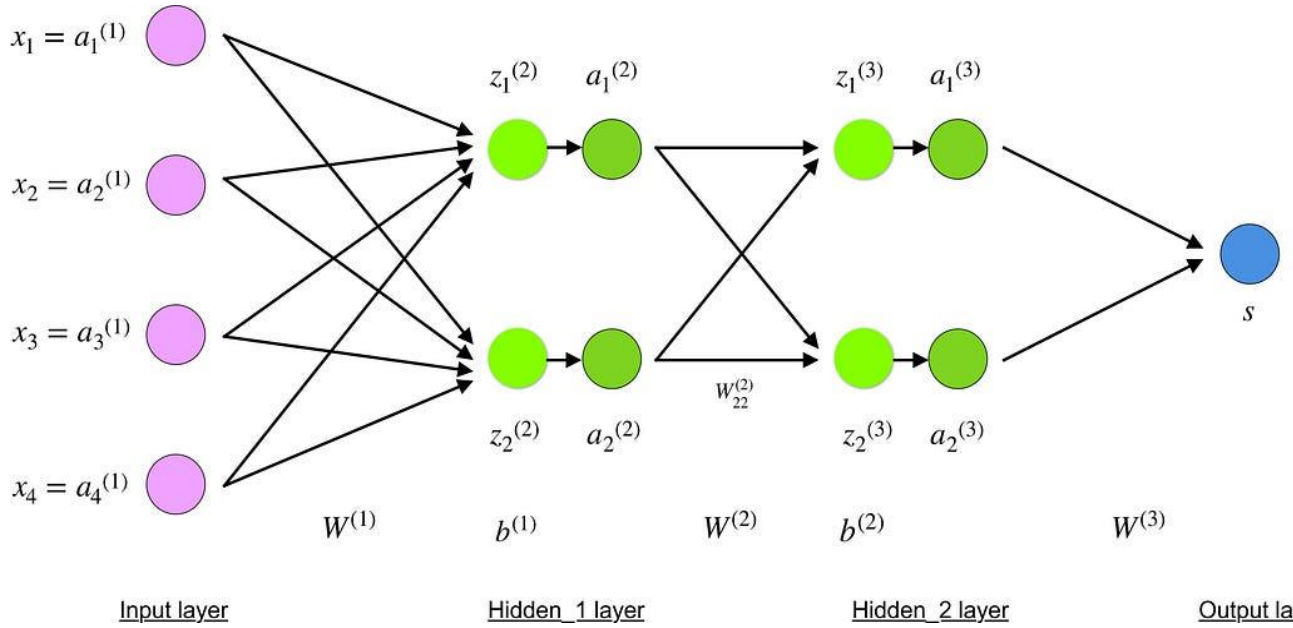
$z^{(2)} = W^{(1)}x + b^{(1)}$ *neuron value at Hidden$_1$ layer*

$a^{(2)} = f(z^{(2)})$ *activation value at Hidden$_1$ layer*

$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$ *neuron value at Hidden$_2$ layer*

$a^{(3)} = f(z^{(3)})$ *activation value at Hidden$_2$ layer*

$s = W^{(3)}a^{(3)}$ *Output layer*



Forward Propagation and evaluation

$x = a^{(1)}$ *Input layer*

$z^{(2)} = W^{(1)}x + b^{(1)}$ *neuron value at Hidden$_1$ layer*

$a^{(2)} = f(z^{(2)})$ *activation value at Hidden$_1$ layer*

$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$ *neuron value at Hidden$_2$ layer*

$a^{(3)} = f(z^{(3)})$ *activation value at Hidden$_2$ layer*

$s = W^{(3)}a^{(3)}$ *Output layer*

- Final step in forward pass is to evaluate the predicted output s against an expected output y.
- Output y is part of the training dataset (x, y) where x: input.
- Evaluation between s and y happens through ***cost function or loss function***. can be MSE (mean squared error) or complex like cross-entropy.
- Equation for cost function C

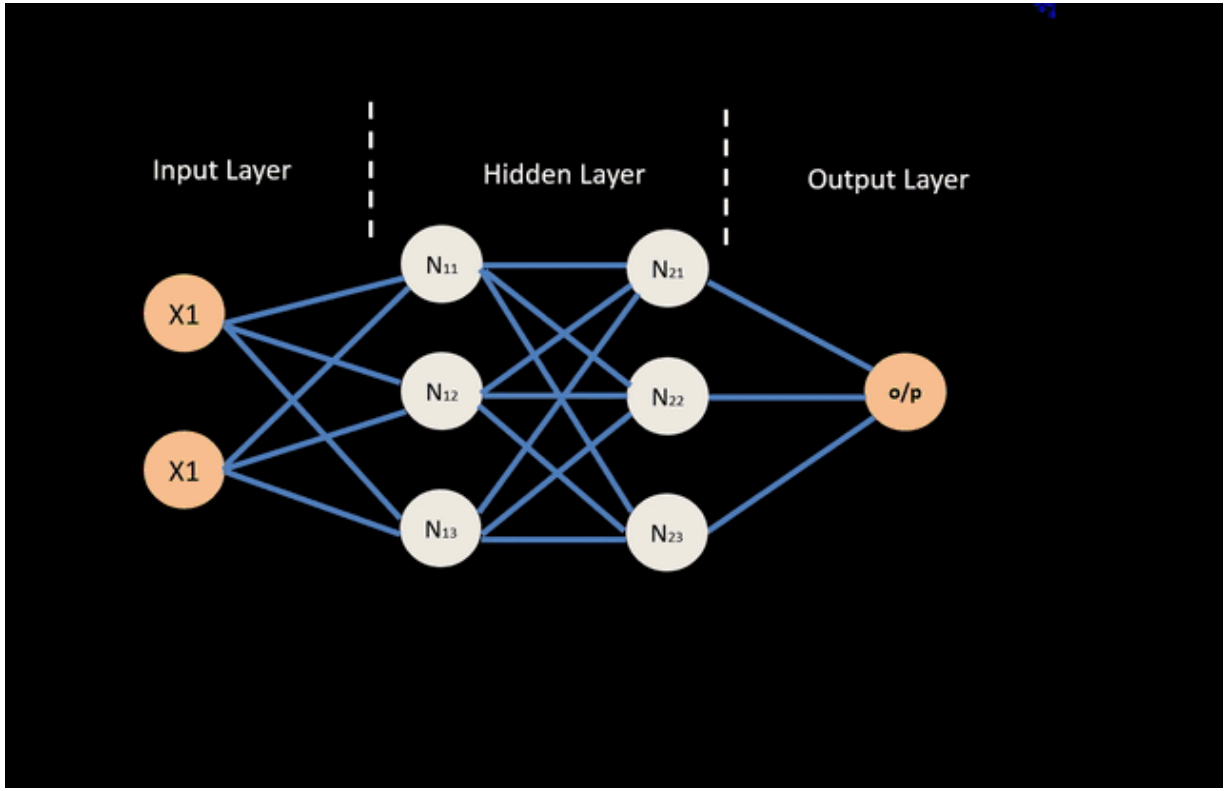
$$C = cost(s, y)$$

- Based on C's value, the model "knows" how much to adjust its parameters in order to get closer to the expected output y. This happens using the ***BACKPROPAGATION ALGORITHM***.

Backpropagation Algorithm

- The algorithm is used to effectively train a neural network through a method called *chain rule*.

- After each *forward pass* through a network, backpropagation performs a *backward pass* while adjusting the model's parameters (*weights and biases*).



Backpropagation and computing gradients

- **Backpropagation aims to minimize the *cost function* by adjusting network's weights and biases.**

- The level of adjustment is determined by the *gradients of the cost function* with respect to those parameters.

why computing gradients?

- Gradient of a function C(x_1, x_2, …, x_m) in point x is a vector of the partial derivatives of C in x.
- Equation for derivative of C in x $\frac{\partial C}{\partial x} = [\frac{\partial C}{\partial x_1}, \frac{\partial C}{\partial x_2}, \ldots, \frac{\partial C}{\partial x_m}]$
- Function C derivative measures the sensitivity to change of function value (output value) with respect to a change in its argument x (input value) or the derivative tells us the direction C is going.
- The gradient shows how much the parameter x needs to change (in positive or negative direction) to minimize C.
- *Compute those gradients happens using a technique called chain rule.*

Using Chain rule

For a single weight $(W_{jk})^l$, the gradient is:

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l} \qquad chain\ rule$$

$$z_j^l = \sum_{k=1}^{m} w_{jk}^l a_k^{l-1} + b_j^l \qquad by\ definition$$

$$m\ -\ number\ of\ neurons\ in\ l-1\ layer$$

$$\frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1} \qquad by\ differentiation\ (calculating\ derivative)$$

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l}a_k^{l-1} \qquad final\ value$$

Equations for derivative of C in a single weight $(W_{jk})^l$

Using Chain rule

Similar set of equations can be applied to $(b\_j)^l$:

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} \qquad chain\ rule$$

$$\frac{\partial z_j^l}{\partial b_j^l} = 1 \qquad by\ differentiation\ (calculating\ derivative)$$

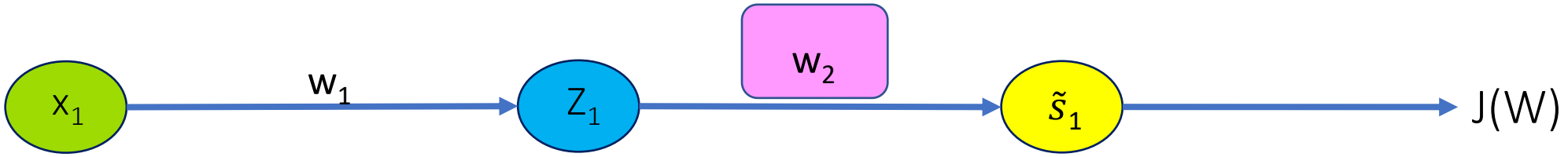
$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l}1 \qquad final\ value$$

Equations for derivative of C in a single bias $(b_j)^l$
The common part in both equations is often called “local gradient” and is expressed as follows:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \qquad local\ gradient$$

Equation for local gradient

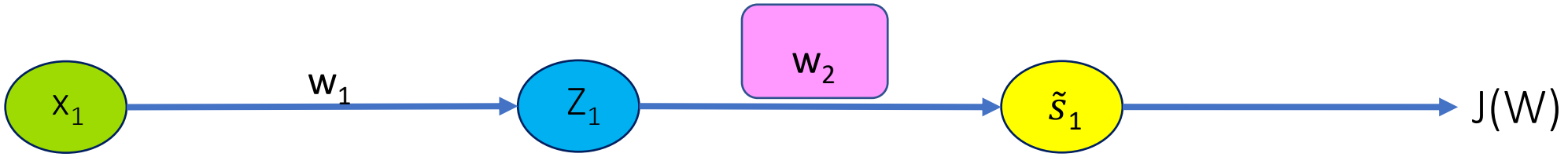
Computing Gradients: Backpropagation



How does a small change in one weight (ex. W_2) affect the final loss **J (W)**?

- Simple network with one input layer, one hidden layer (one hidden neuron) and one output layer.
- Computing the gradient of loss of J(W) with respect to w2, can perform lot of changes in loss value.
- How does a small change in one weight (ex. W2) affect the final loss J (W)?

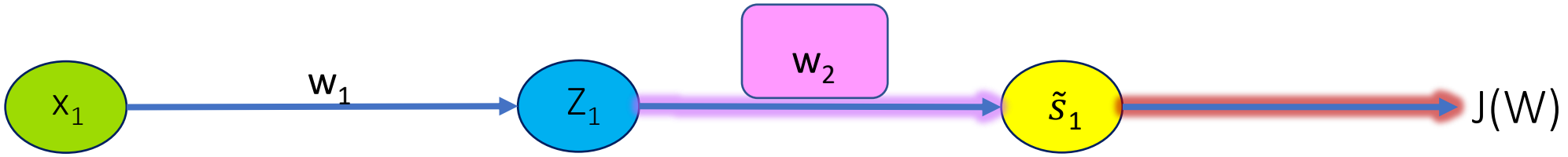
Computing Gradients: Backpropagation



Gradient loss of W with respect to w_2 $\frac{\partial J(W)}{\partial w_2}$

Let's use the chain rule!

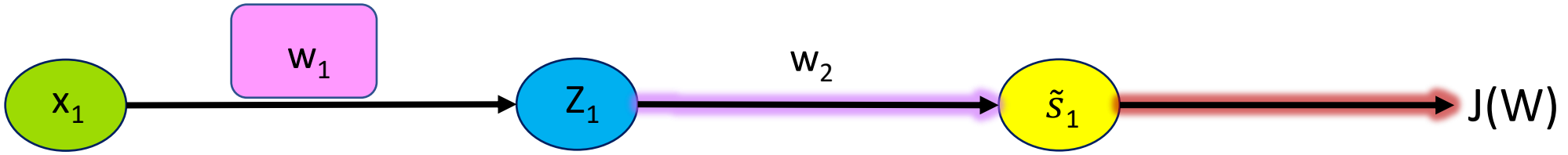
Computing Gradients: Backpropagation



Split this into the gradient of our loss with respect to output.

$$\frac{\partial J(W)}{\partial (w_2)} = \frac{\partial J(W)}{\partial \tilde{s}} * \frac{\partial \tilde{s}}{\partial (w_2)}$$

Computing Gradients: Backpropagation



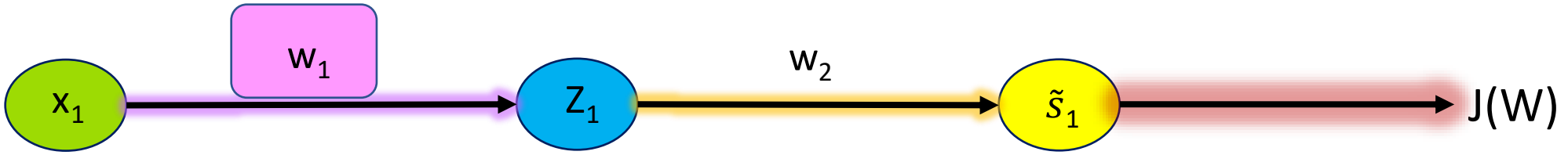
- Replace w2 with w1 and apply chain rule but we now notice that the gradient of output s with respect to w1 is not directly computable , apply chain rule again to evaluate.

$$\frac{\partial J(W)}{\partial (w_1)} = \frac{\partial J(W)}{\partial \tilde{s}} * \frac{\partial \tilde{s}}{\partial (w_1)}$$

Apply chain rule!

Apply chain rule!

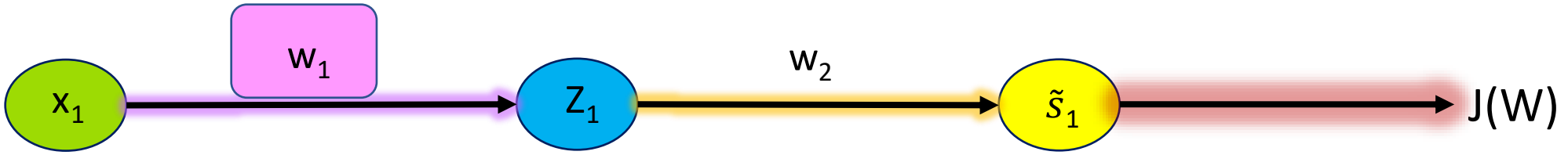
Computing Gradients: Backpropagation



- Apply the chain rule, and split with respect to Z.
- Back propagation is done performing it, all the gradients from the output to the input that allow the error to propagate from output layer to input layer.

$$\frac{\partial J(W)}{\partial(w_1)} = \frac{\partial J(W)}{\partial \tilde{s}} * \frac{\partial \tilde{s}}{\partial(Z_1)} * \frac{\partial Z_1}{\partial w_1}$$

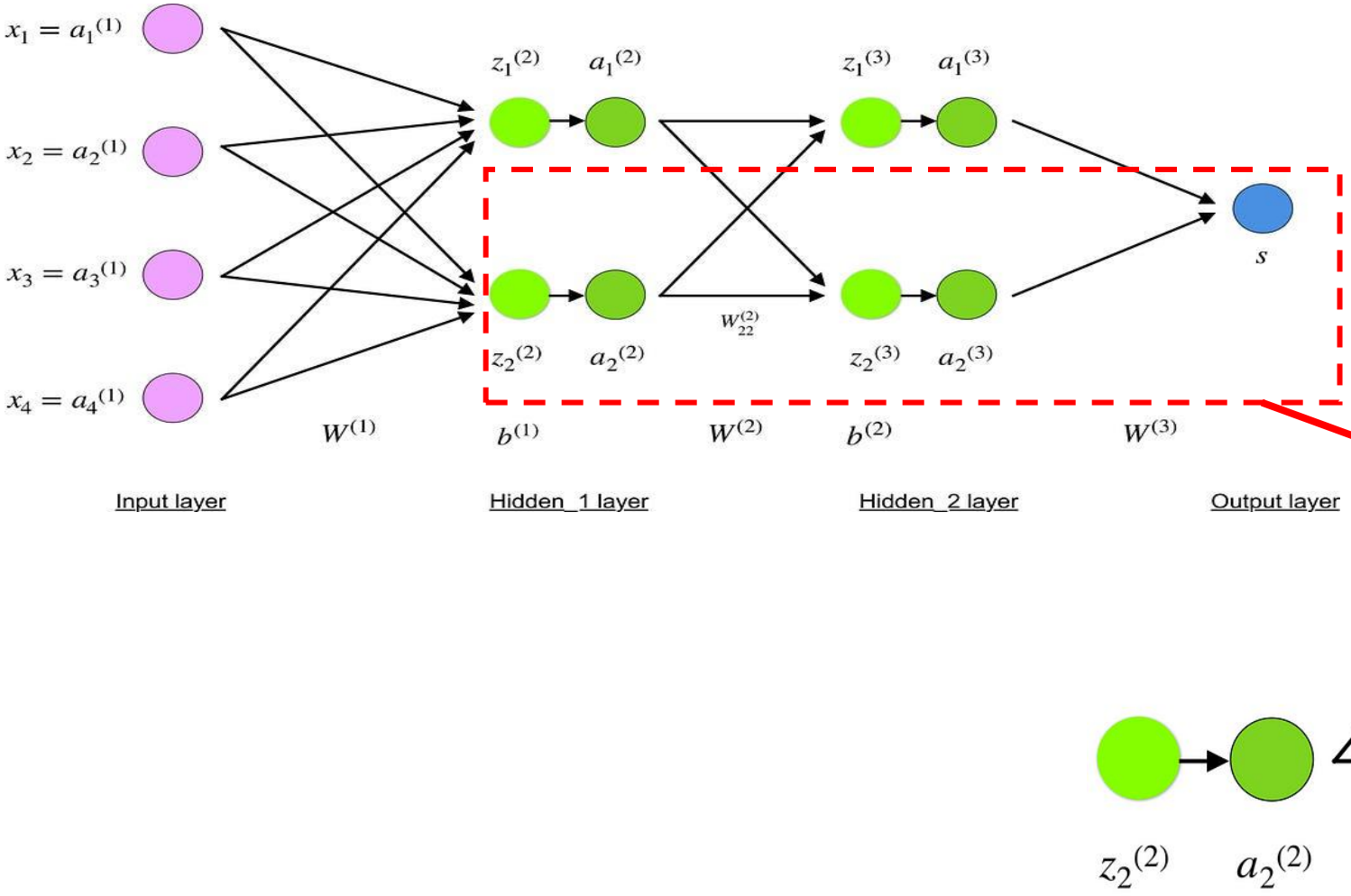
Computing Gradients: Backpropagation

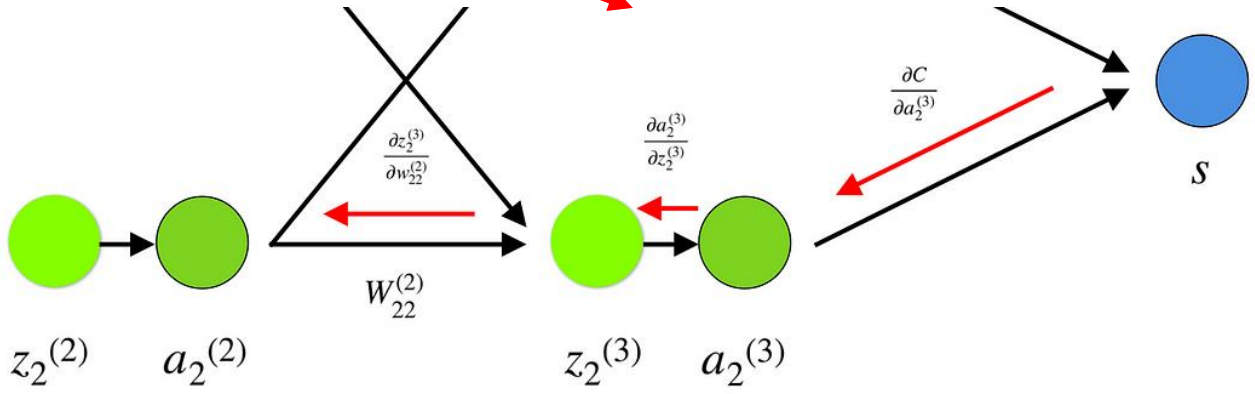


- Repeat this for every weight in the network using gradients from later layers

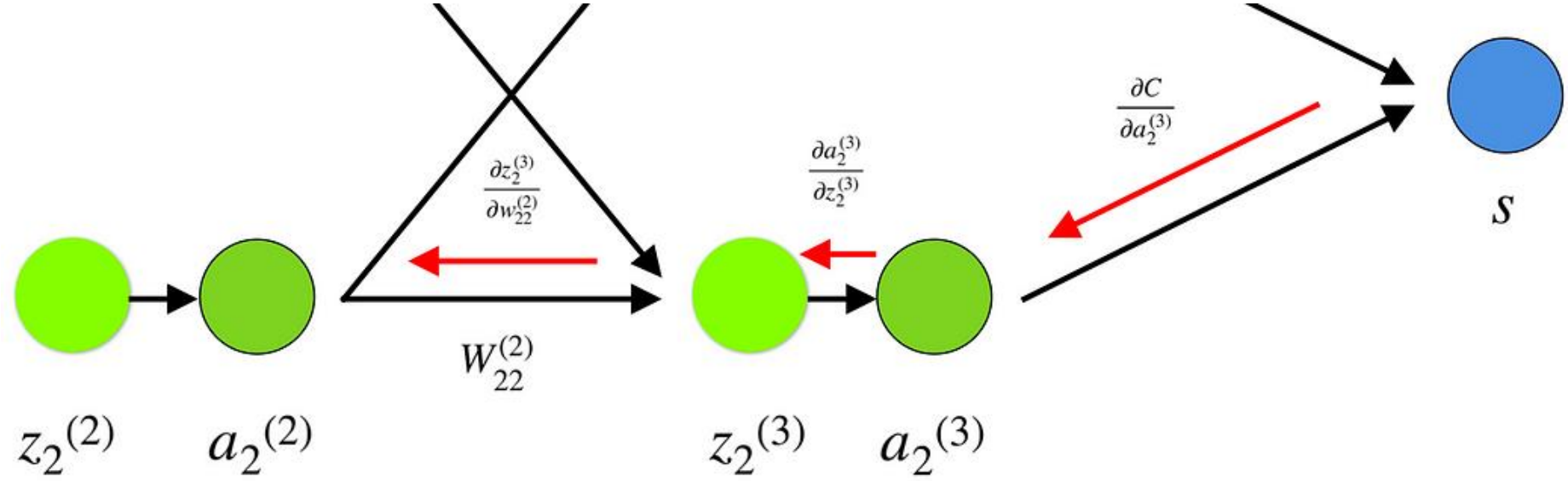
$$\frac{\partial J(W)}{\partial (w_1)} = \frac{\partial J(W)}{\partial \tilde{s}} * \frac{\partial \tilde{s}}{\partial (Z_1)} * \frac{\partial Z_1}{\partial w_1}$$

Calculate the gradient of C with respect to a single weight $(W_{22})^2$?





Calculate the gradient of C with respect to a single weight $(W_{22})^2$?



- Weight $(W_{22})^2$ connects $(a_2)^2$ and $(Z_2)^2$, so computing the gradient requires applying the chain rule through $(Z_2)^3$ and $(a_2)^3$:

$$\frac{\partial C}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial z_2^{(3)}} \cdot \frac{\partial z_2^{(3)}}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial a_2^{(3)}} \cdot \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \cdot a_2^{(2)} = \frac{\partial C}{\partial a_2^{(3)}} \cdot f'(z_2^{(3)}) \cdot a_2^{(2)}$$

- Equation for derivative of C in $(W_{22})^2$
- Calculating the final value of derivative of *C* in $(a_2)^3$ requires knowledge of the function *C*. Since *C* is dependent on $(a_2)^3$,