Logistic Regression

Logistic regression:

"logit regression," is a **supervised learning** algorithm primarily **used for binary classification(Yes or No) tasks.**

It is commonly employed to determine whether an input belongs to one class or another, such as deciding whether an image is a cat or not a cat.

Logistic regression **predicts the *probability*** that an input can be **categorized into a single primary class**.

It is commonly used to group outputs into two categories**: the primary class** and **not the primary class.**

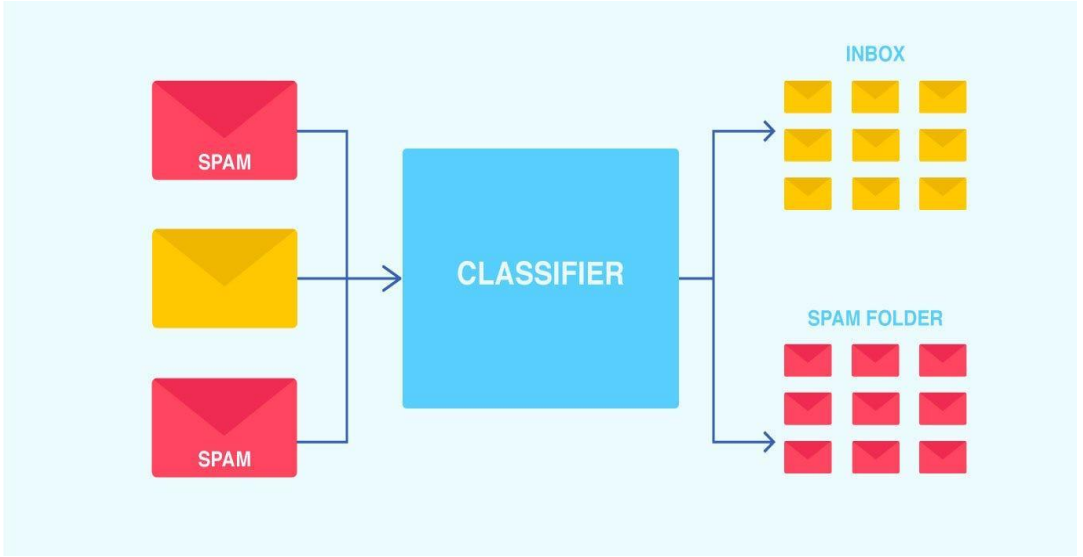
logistic regression is typically used for **binary categorization** rather than **predictive modeling**

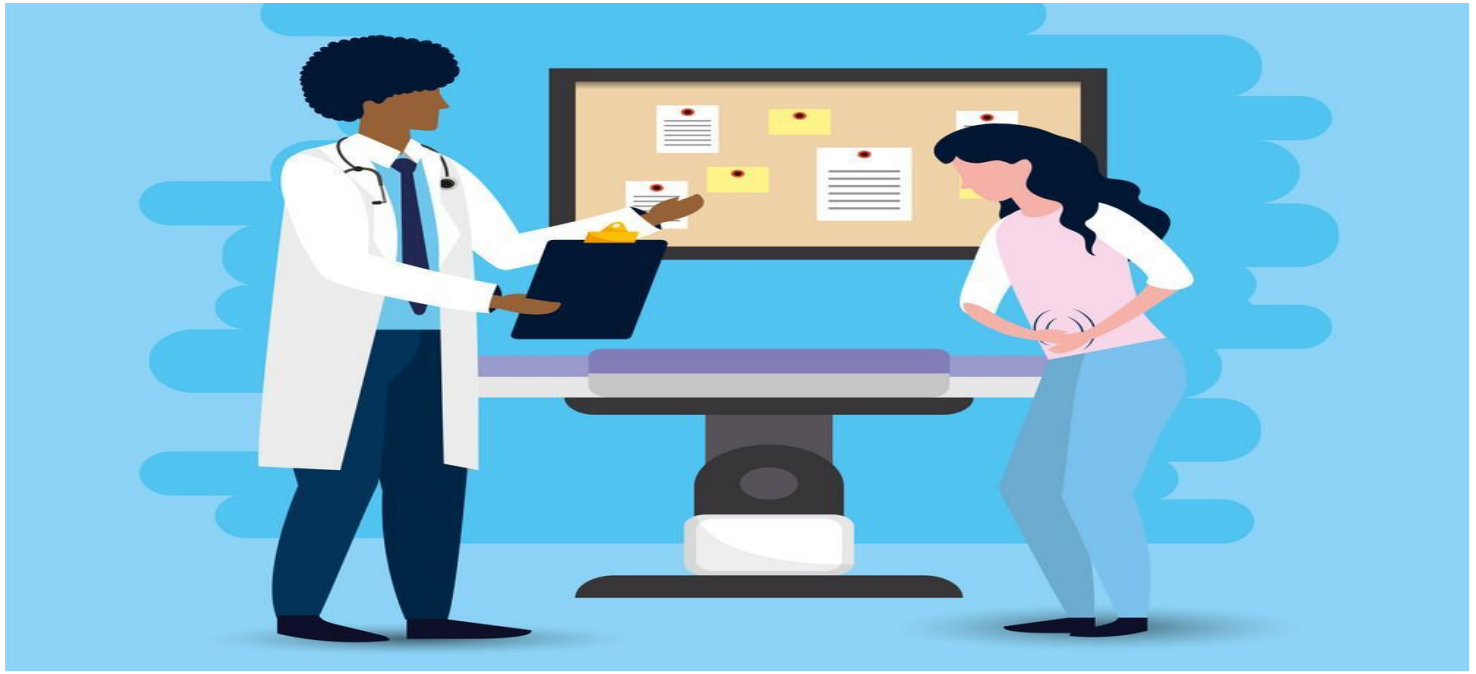
It creates a **threshold or boundary for binary classification**.

For example: Any output value between **0 and 0.49** might be **classified as one group**,
while values between **0.50 and 1.00** would be **classified as the other group**.

Application: image recognition, spam email detection, or medical diagnosis where we need to categorize data into distinct classes.

Applications:





Logistic regression

- Name is somewhat misleading. Really a technique for classification, not regression.
  - "Regression" comes from fact that we fit a linear model to the feature space.
- Involves a more probabilistic view of classification.

Examples

use *credit score* and *bank balance* to predict whether or not a given customer will **default on a loan.** (Response variable = “Default” or “No default”)

use *square footage* and *number of bathrooms* to predict whether or not a house in a certain city will be listed at a selling price of $200k or more. (Response variable = “Yes” or “No”)

Binary logistic regression expression

$$Y = b_o + b_1X_1 + b_2X_2 + \ldots\ldots\ldots\ldots b_kX_k + E$$

BINARY

Y = Dependent Variables

ß$_o$ = Constant

ß$_1$ = Coefficient of variable X_1

X_1 = Independent Variables

E = Error Term

Logistic regression cont…

So what shall we do?

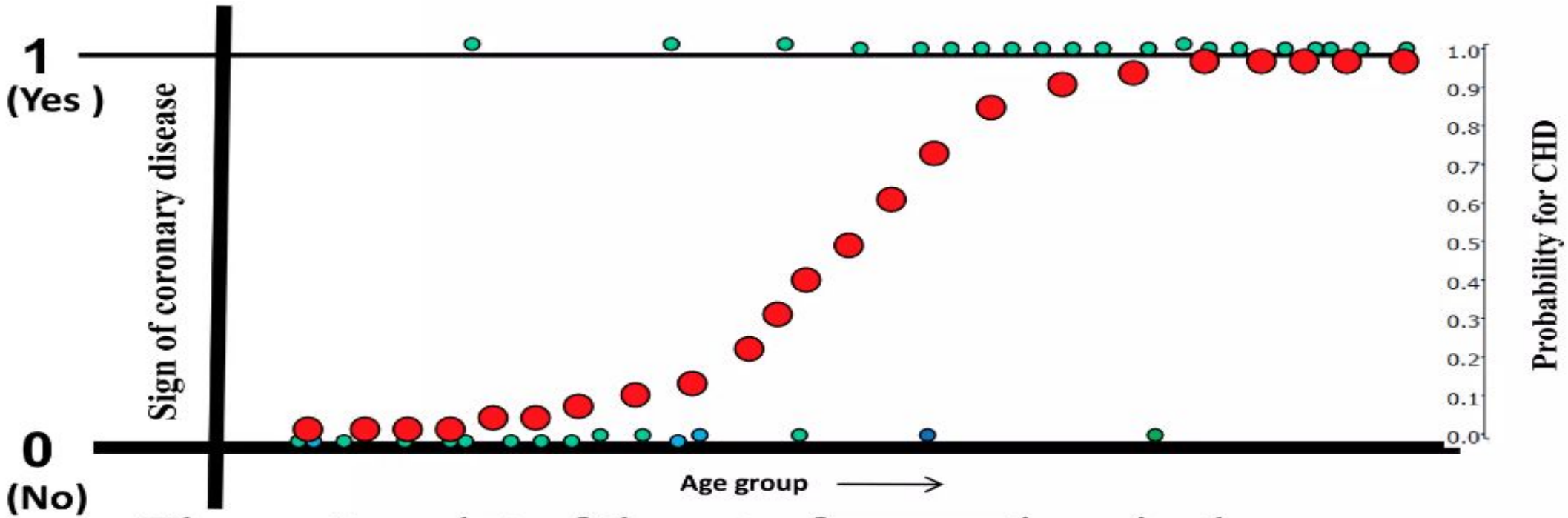
✓Rather than dealing with a **single age** data with binary outcomes, let us **group the age data** so that we can get proportions (probabilities) of success (1s) at different age groups

✓In doing so, we can get intermediate proportions between 0 and 1

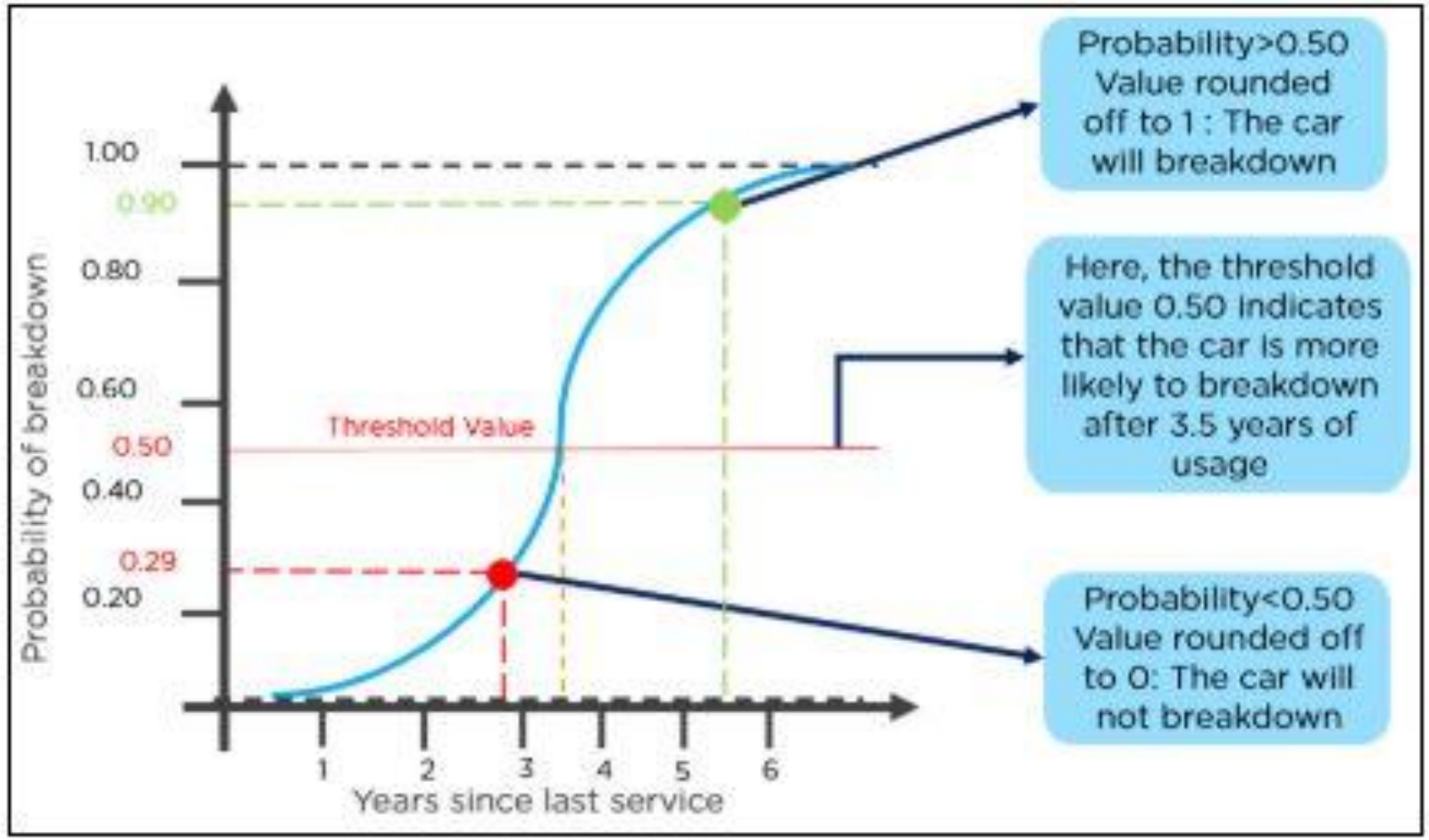
		Diseased	
Age group	# in group	#	probability
20 - 29	5	0	0
30 - 39	6	1	0.17
40 - 49	7	2	0.29
50 - 59	7	4	0.57
60 - 69	5	4	0.80
70 - 79	2	2	1.00
80 - 89	1	1	1.00

Logistic regression cont…

✓The probabilities in the above table are the same as the proportions of individuals with CHD in each age category.



The scatter plot of the set of proportions in the age ranges could give us the above S-shaped curve (red color)


Probability>0.50
Value rounded off to 1 : The car will breakdown
Here, the threshold value 0.50 indicates that the car is more likely to breakdown after 3.5 years of usage
Probability<0.50
Value rounded off to 0: The car will not breakdown
Probability of breakdown
1.00
0.90
0.80
0.60
0.50
0.40
0.29
0.20
Threshold Value
1
2
3
4
5
6
Years since last service

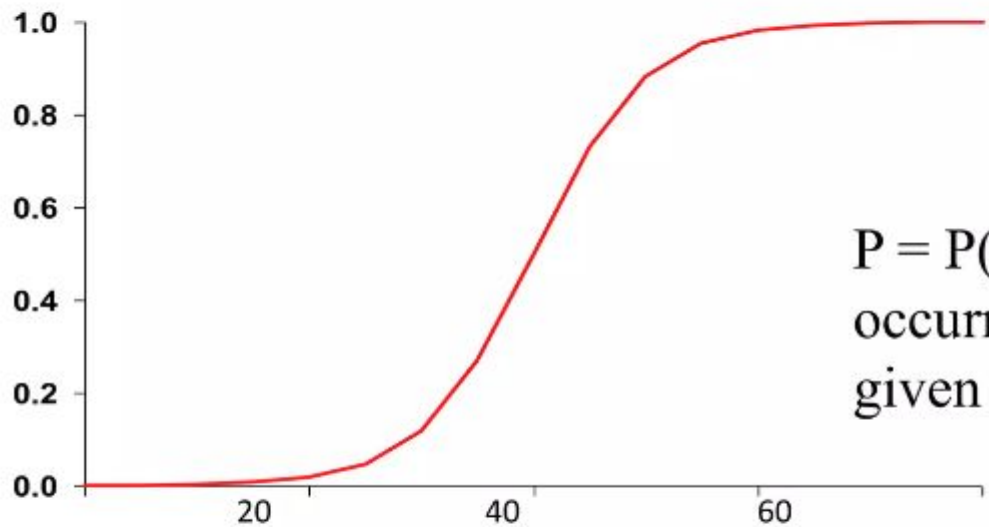
Logistic regression cont…

- Again such S-shaped (sigmoidal) curve is difficult to describe with a linear equation for two reasons.
- First, even though it seems linear at the center of the curve, the extremes do not follow a linear trend;
- Second, the errors are neither normally distributed nor constant across the entire range of data

Question! So what do we do with this S-Shaped curve?

- **Answer:**
- **First: Find a function that best fits (be linked) with this S-shaped graph**
- **Second: Find another function that transforms the S-shaped graph into linear function**

(I) Finding a function that best fits with the S- shaped graph of probability



$$P = \frac{e^{\alpha+\beta x}}{1 + e^{\alpha+\beta x}}$$

P = P(y/x) = P(success given x occurred) = P(a person is +ve CHD given his age is x)

✓We call the above mathematical expression a **logistic function**

✓It always has an S- shaped curve within the range of 0 and 1 for any x

✓That is why we linked it with p (probability) which has the same S-shape in the same range of 0 to 1

(II) Transforming to linear function using logit function

$$P = \frac{e^{\alpha+\beta x}}{1 + e^{\alpha+\beta x}}$$

This is the logistic function

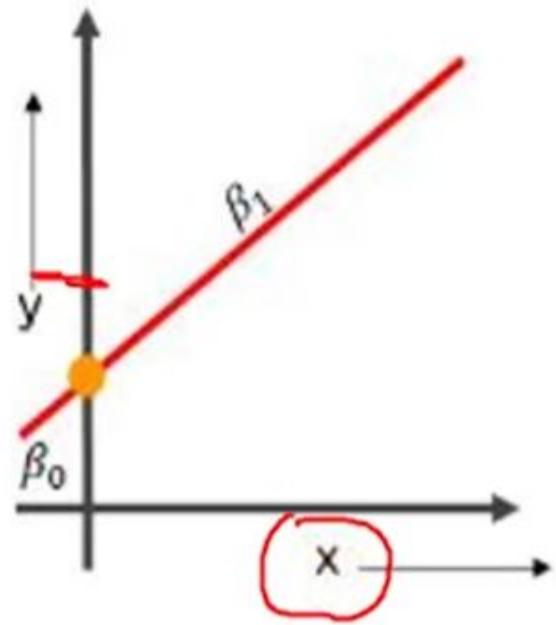
$$\frac{P}{1-P} = e^{\alpha+\beta X}$$

The odds of an event

$$\underbrace{\ln\left(\frac{P}{1-P}\right)}_{\text{logit of } P} = \alpha + \beta X$$

- ✓α = log odds of event in unexposed
- ✓β = log Odds Ratio associated with being exposed
- ✓e^{β} = Odds Ratio

- To understand logistic regression, let's go over the odds of success.
- $Odds\ (\theta) = \frac{Probability\ of\ an\ event\ happening}{Probability\ of\ an\ event\ not\ happening}$
- $Odds\ (\theta) = \frac{p}{1-p}$
- The values of odds range from **zero to ∞** and the values of probability lies between **zero and one**.
- Consider the equation of a straight line:
- $\boldsymbol{y = \beta 0 + \beta 1 * x}$



Logistic Regression – Algorithm & Solved Example

- Now to predict the odds of success, we take log on odds formula:

$$\log\left(\frac{p(x)}{1-P(x)}\right) = \beta_0 + \beta_1 x$$

- Exponentiating both the sides, we have:

$$e^{\ln(x)} = x$$

$$e^{ln}\left(\frac{p(x)}{1-p(x)}\right) = e^{\beta_0+\beta_1 x}$$

$$\left(\frac{p(x)}{1-p(x)}\right) = e^{\beta_0+\beta_1 x}$$

- $Let\ Y = e^{\beta 0+\beta 1 * x}$
- $Then \quad \frac{p(x)}{1-p(x)} = Y$
- $p(x) = Y(1 - p(x))$
- $p(x) = Y - Y(p(x))$
- $p(x) + Y(p(x)) = Y$
- $p(x)(1+Y) = Y$
- $p(x) = \frac{Y}{1+Y}$

$$p(x) = \frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}}$$

The equation of the sigmoid function is:

$$p(x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x)}}$$

- The student dataset has entrance mark based on the historic data of those who are selected or not selected.
- Based on the logistic regression, the values of the learnt parameters are $\beta_0 = 1 \; and \; \beta_1 = 8.$
- Assuming marks of x = 60, compute the resultant class.

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

$$\beta_0 + \beta_1 x = 481$$

$$p(x) = \frac{1}{1 + e^{-481}} = 0.44$$

- If we assume the threshold value as 0.5, then it is observed that 0.44 < 0.5, therefore, the candidate with marks 60 is not selected.

Question: Practice Problem 6 a. Construct a logistic regression model with two predictors for the riding mower example with β,--25.9382, β1-0.1 109, β,-0.96...

Practice Problem 6

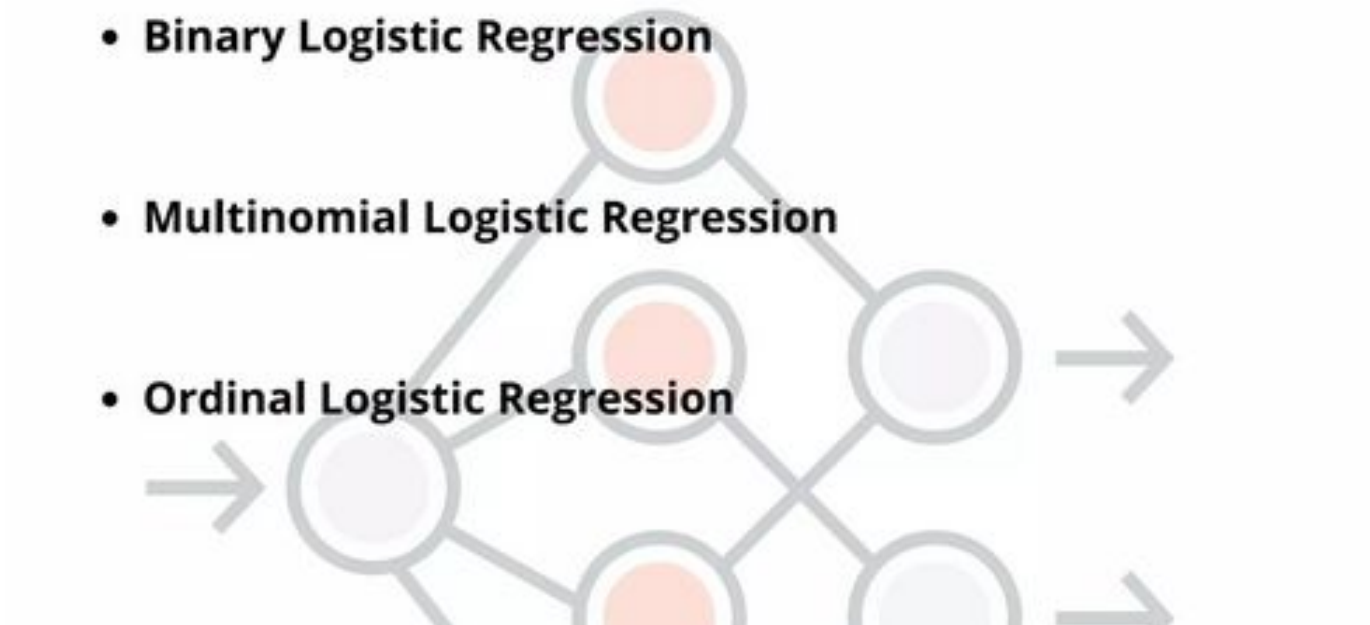
a. Construct a logistic regression model with two predictors for the riding mower example with $\beta_0 = -25.9382, \beta_1 = 0.1109, \beta_2 = 0.9638$, where β_1 and β_2 are for the "Income" and "Lot_Size" variables, respectively.

b. Using the logistic regression model with probability cutoff = 0.75, classify the following 6 customers as "Owner" or "Nonowner" : if $p \geq 0.75$ then the case as a "Owner". Present the results in a classification matrix.

Customer#	Income	Lot_Size	Ownership
1	60.0	18.4	Owner
2	64.8	21.6	Owner
3	84.0	17.6	Nonowner
4	59.4	16.0	Nonowner
5	108.0	17.6	Owner
6	75	19.6	Nonowner

Types of Logistic Regression

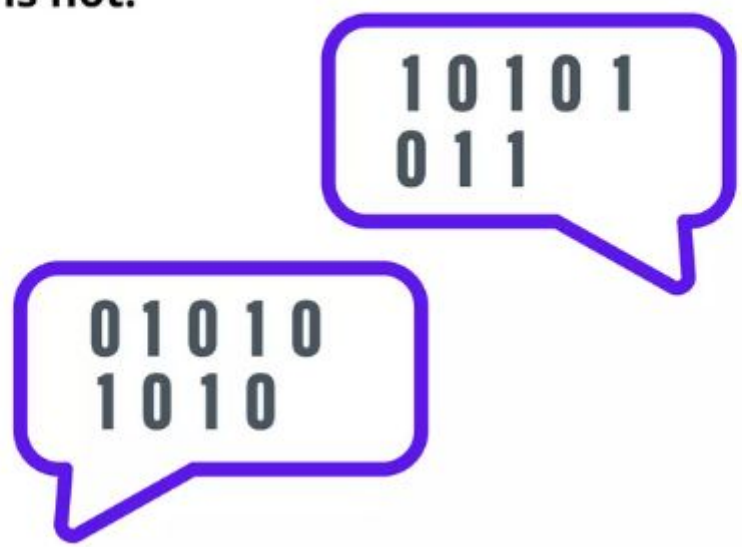
- Binary Logistic Regression
- Multinomial Logistic Regression
- Ordinal Logistic Regression



Binary Logistic Regression

- Based on the values of the independent variables, binary logistic regression is used to estimate the likelihood of being a case (predictors).

- The odds are calculated by dividing the chance that a given result is a case by the probability that it is not.



Multinomial Logistic Regression

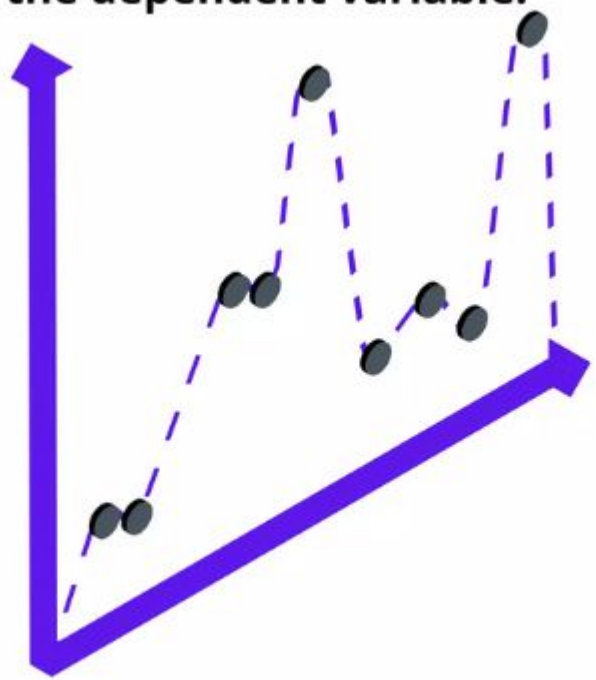
- Multinomial logistic regression is a classification technique that extends logistic regression to situations with more than two discrete outcomes.

- Three or more categories without ordering. Example: Predicting which food is preferred more (Veg, Non-Veg, Vegan)

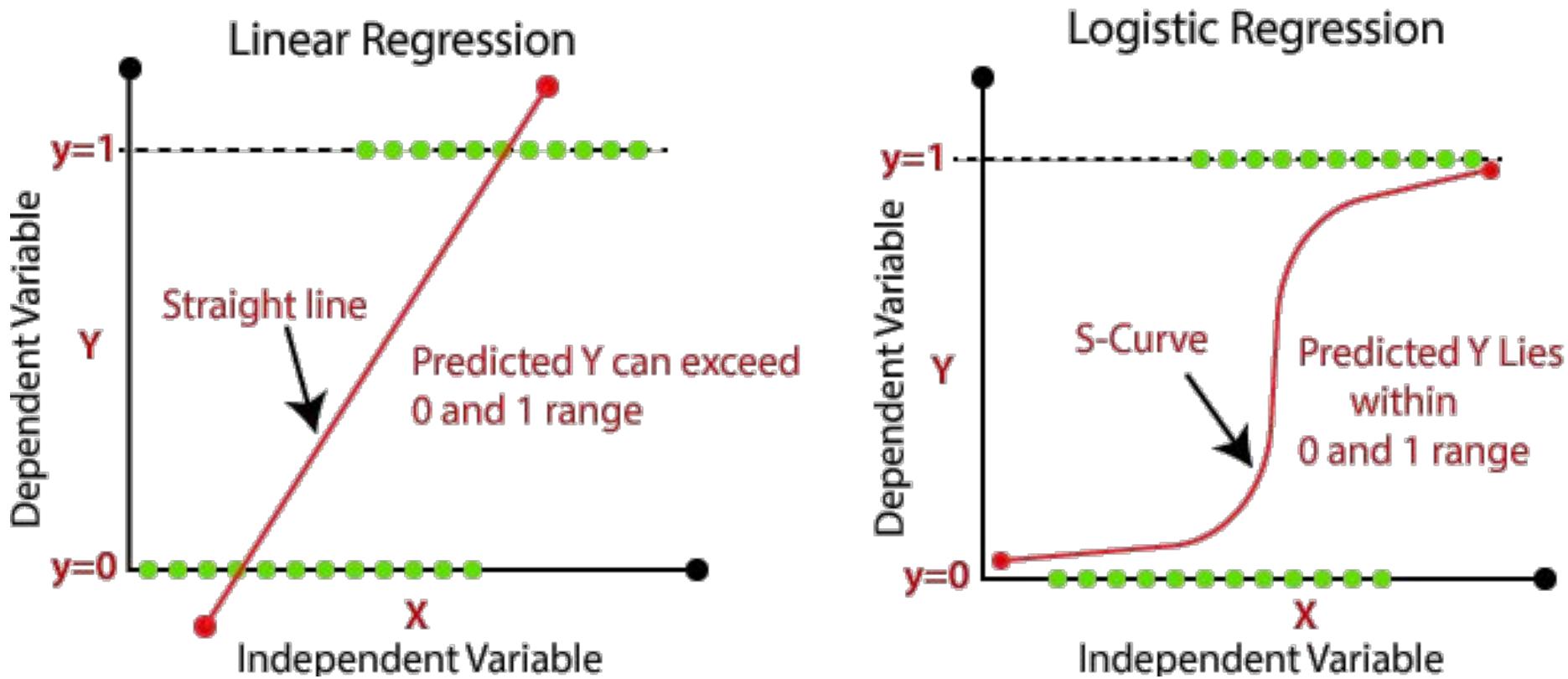
Ordinal Logistic Regression

- Ordinal Regression (sometimes called Ordinal Logistic Regression) is a binomial logistic regression extension.

- With 'ordered' multiple categories and independent variables, ordinal regression is used to predict the dependent variable.



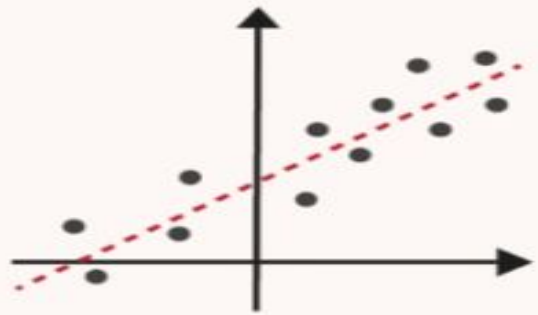
Basis	Linear Regression	Logistic Regression
Core Concept	The data is modelled using a straight line	The probability of some obtained event is represented as a linear function of a combination of predictor variables.
Used with	Continuous Variable	Categorical Variable
Output/Prediction	Value of the variable	Probability of occurrence of event
Accuracy and Goodness of fit	measured by loss, R squared, Adjusted R squared etc.	Accuracy, Precision, Recall, F1 score, ROC curve, Confusion Matrix, etc


Linear Regression
y=1
y=0
Y
Dependent Variable
Straight line
Predicted Y can exceed
0 and 1 range
X
Independent Variable
Logistic Regression
y=1
y=0
Y
Dependent Variable
S-Curve
Predicted Y Lies
within
0 and 1 range
X
Independent Variable

Linear regression VS Logistic regression

Linear regression

- Econometric modeling
- Marketing mix model
- Customer lifetime value



Continuous > Continuous

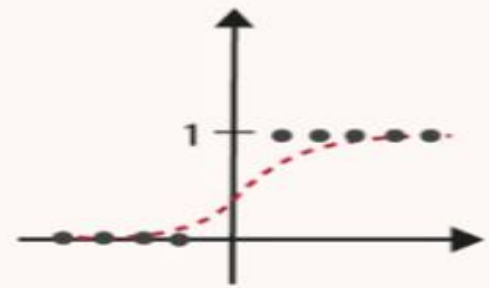
$$y = a_0 + \sum_{i=1}^{N} a_i x_i$$

Im(y-x1 + x2, data)

1 unit increase in
x increases y by a

Logistic regression

- Customer choice model
- Click-through rate
- Conversion rate
- Credit scoring



Continuous > True/False

$$y = \frac{1}{1 + e^{-z}}$$

$$z = a_0 + \sum_{i=1}^{N} a_i x_i$$

glm(y-x1 + x2, data), family = binomial())

1 unit increase in x
increases log odds by a

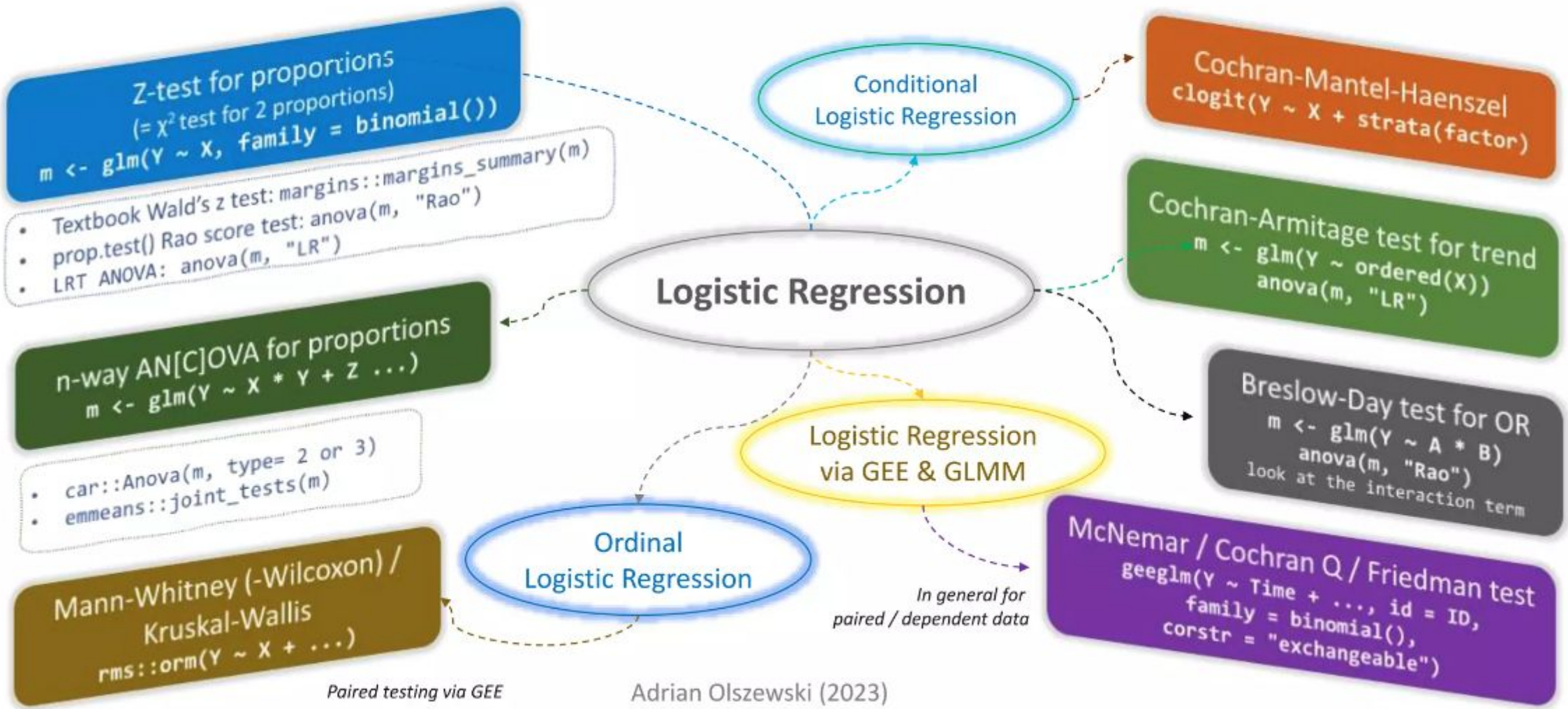
LOGISTIC REGRESSION

Advantages and Disadvantages of Logistic Regression

Advantages	Disadvantages
Logistic regression works well for cases where the dataset is linearly separable.	The major drawback is, it is vulnerable to overfitting.
Logistic regression is much easier to implement than other methods, especially in the context of machine learning.	It can not handle many categorical variables, which means it cannot be used on a heavy model.
Logistic regression provides useful insights.	If independent variables are not correlated with the target variable, then this technique does not work properly.

Testing hypotheses about binary outcomes (proportions)

with the Logistic Regression & selected friends



Adrian Olszewski (2023)

Characteristics of the logistic function

The S shaped curve of logistic function has the following characteristics:

Function: $$P = \frac{e^{\alpha+\beta x}}{1 + e^{\alpha+\beta x}}$$

If β is the slope of the linear function after logit transformation then,

- ✓ The S-shaped curve has a slope equal to p(1-p) β , where p is the probability at X = *x*
- ✓ As we move to the two extremes of x or p, the slope closes to 0
- ✓ The (x, p) coordinate on which the slope reaches its pick is (-α/β, 0.5). The value of x at this point is called *median effective level* denoted by EL_{50}

Logistic regression cont…

Example on the data given above:

➢The analysis of logistic regression is computer intensive

➢After entering the above data using SPSS and running it for binary logistic regression, the following result has been obtained.

Variable	β	S.E	Wald	Sig.	Exp(β)	95.0% C.I. for EXP(β)	
						Lower	Upper
age	0.132	0.046	8.053	0.005	1.141	1.042	1.249
Constant	-6.708	2.354	8.121	0.004	0.001		

➢For a unit increase in age of a person, the odds of being positive for CHD increases by a factor of 1.141

➢The 95% CI for this estimate (i.e. Odds Ratio) is (1.04, 1.23)

Logistic Regression cont…

Example on categorical variable (residence) Vs patient satisfaction on service

	Patient satisfaction		
Residence	**Unsatisfied**	**Satisfied**	**Total**
Rural	98	17	115
Urban	205	154	359
Total	303	171	474

- ✓ Odds for Rural: $\frac{p}{(1-p)} = \frac{0.85}{0.15} = 5.76$ ⟹ ln Odds: $\ln(5.76) = 1.75$
- ✓ Odds for Urban: $\frac{p}{(1-p)} = \frac{0.571}{0.429} = 1.33$ ⟹ ln Odds: $\ln(1.33) = 0.285$
- ✓ Odds Ratio = $\frac{5.76}{1.33} = 4.33$

Δ ln Odds = ln OR = β= 1.47

↓

OR = $e^{1.47} = 4.33$

- ✓ OR remains the same by the two methods

CLUSTERING

CLUSTERING TECHNIQUES

Partitional clustering decomposes a data set into a set of disjoint clusters.

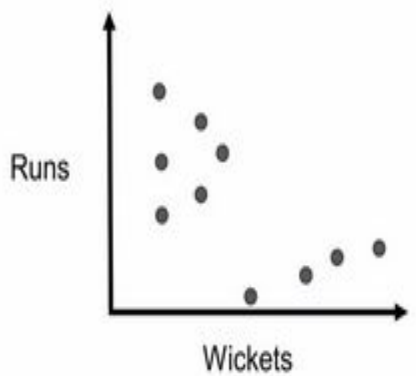
Partitional clustering (or partitioning clustering) are clustering method used to classify observations, within a data set, into multiple groups based on their similarity. The algorithms require the analyst to specify the number of clusters to be generated ($N \geq K$). This course describes the commonly used partitional, including: k means clustering

What is K-Means Clustering?

Assign data points

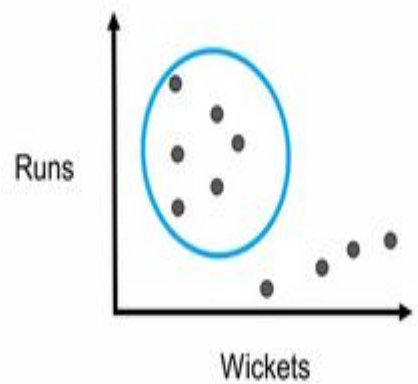
Here, we have our dataset with x and y coordinates

Now, we want to cluster this data using **K-Means**



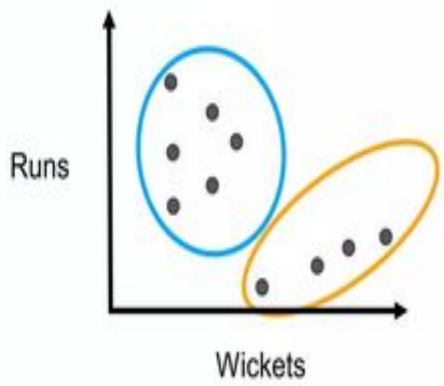
Cluster 1

We can see that this cluster has players with high runs and low wickets

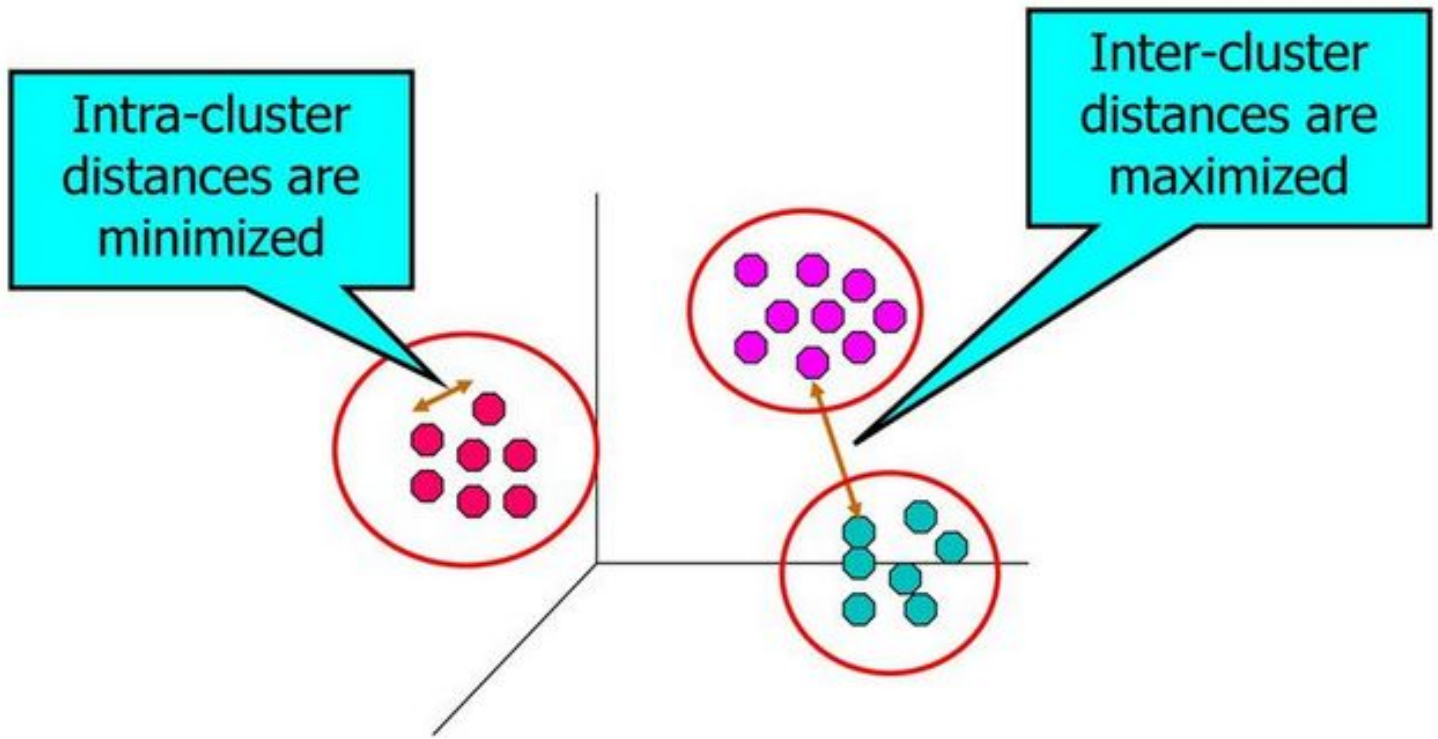


Cluster 2

And here, we can see that this cluster has players with high wickets and low wickets



- Finding groups of objects such that the objects in a group will be similar (or related) to one another and different from (or unrelated to) the objects in other groups

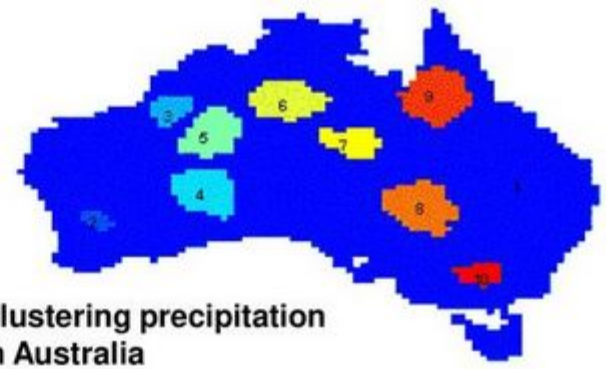


Applications of Cluster Analysis

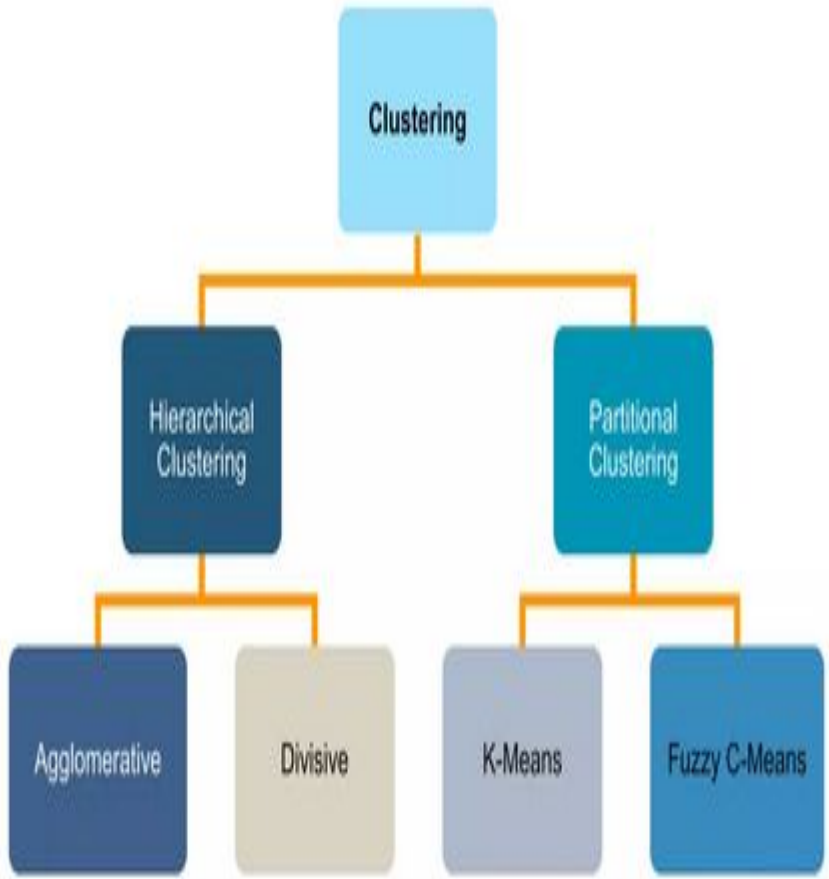
- **Understanding**
  - Group related documents for browsing, group genes and proteins that have similar functionality, or group stocks with similar price fluctuations

	Discovered Clusters	*Industry Group*
1	Applied-Matl-DOWN,Bay-Network-Down,3-COM-DOWN, Cabletron-Sys-DOWN,CISCO-DOWN,HP-DOWN, DSC-Comm-DOWN,INTEL-DOWN,LSI-Logic-DOWN, Micron-Tech-DOWN,Texas-Inst-Down,Tellabs-Inc-Down, Natl-Semiconduct-DOWN,Oracl-DOWN,SGI-DOWN, Sun-DOWN	Technology1-DOWN
2	Apple-Comp-DOWN,Autodesk-DOWN,DEC-DOWN, ADV-Micro-Device-DOWN,Andrew-Corp-DOWN, Computer-Assoc-DOWN,Circuit-City-DOWN, Compaq-DOWN, EMC-Corp-DOWN, Gen-Inst-DOWN, Motorola-DOWN,Microsoft-DOWN,Scientific-Atl-DOWN	Technology2-DOWN
3	Fannie-Mae-DOWN,Fed-Home-Loan-DOWN, MBNA-Corp-DOWN,Morgan-Stanley-DOWN	Financial-DOWN
4	Baker-Hughes-UP,Dresser-Inds-UP,Halliburton-HLD-UP, Louisiana-Land-UP,Phillips-Petro-UP,Unocal-UP, Schlumberger-UP	Oil-UP

- **Summarization**
  - Reduce the size of large data sets



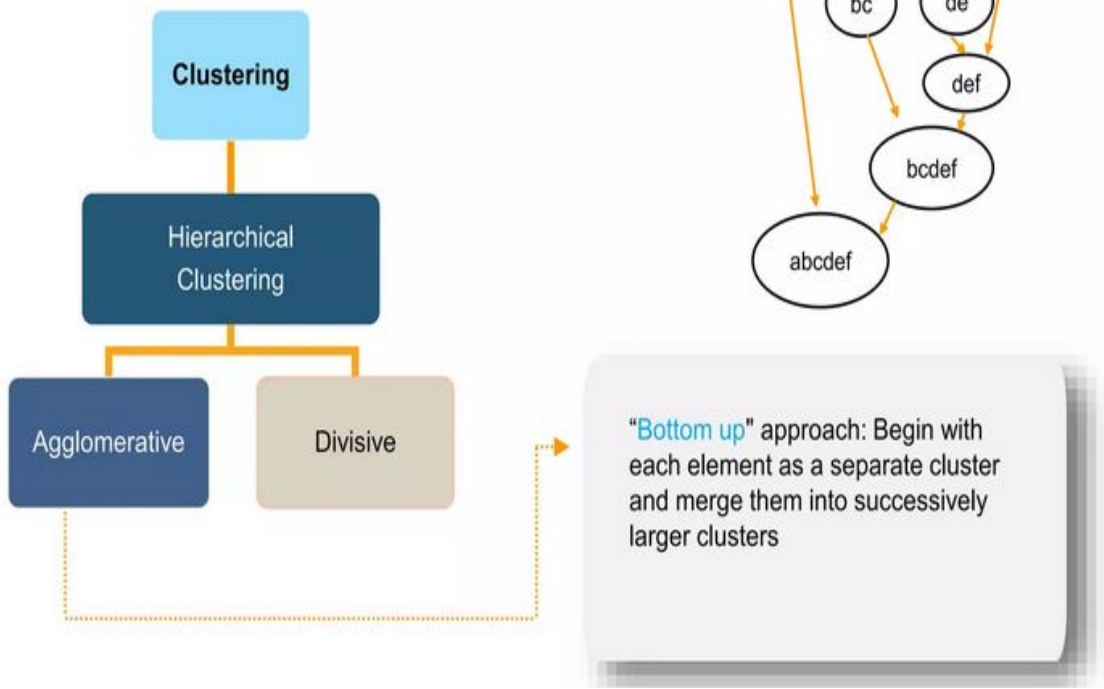
Clustering precipitation in Australia



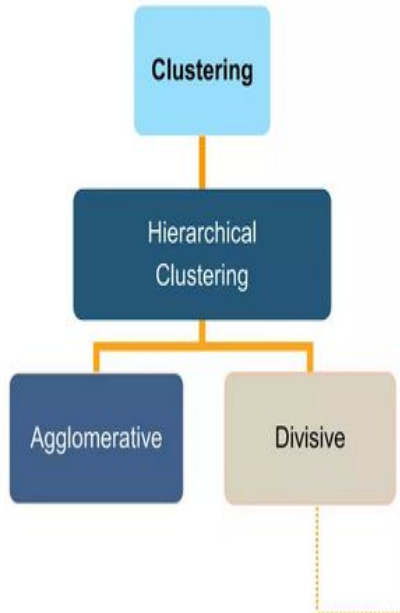
Types of Clusterings

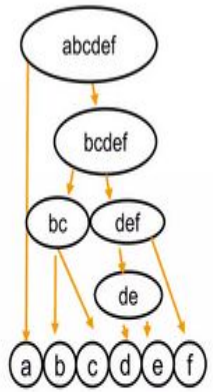
- A clustering is a set of clusters
- Important distinction between hierarchical and partitional sets of clusters
- Partitional Clustering
  - A division data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset
- Hierarchical clustering
  - A set of nested clusters organized as a hierarchical tree

Types of Clustering



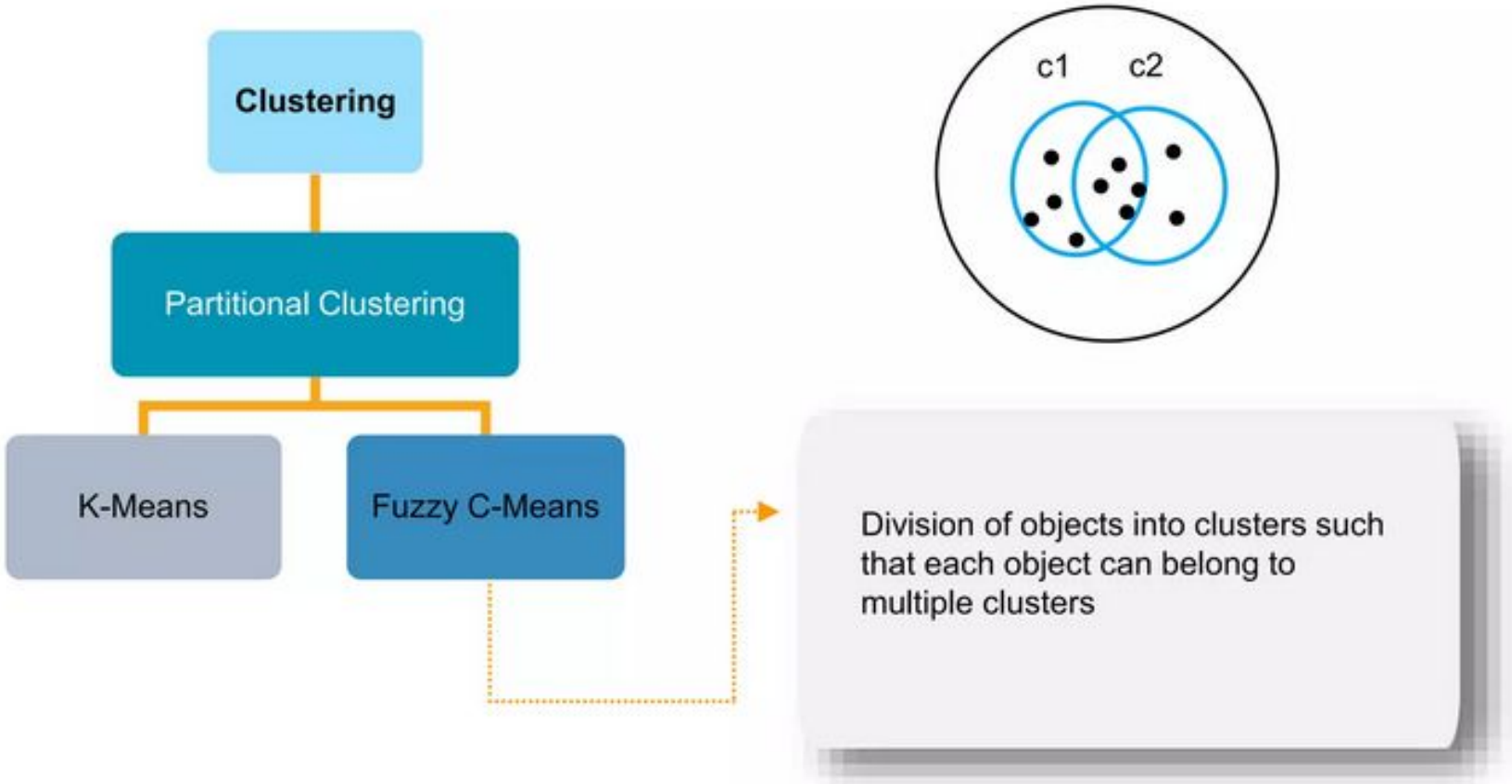
"Bottom up" approach: Begin with each element as a separate cluster and merge them into successively larger clusters



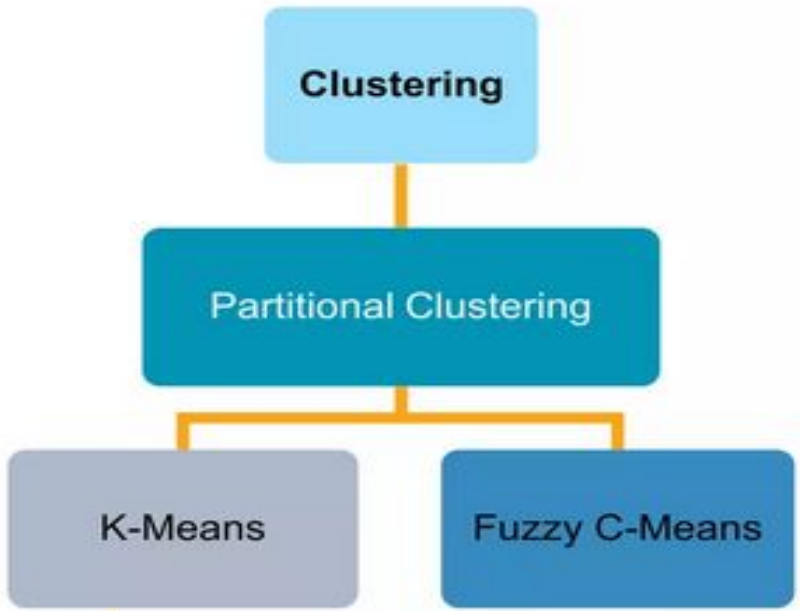


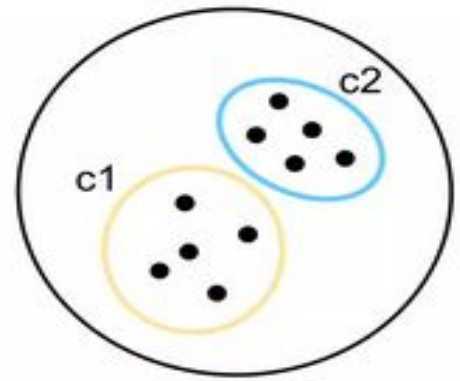
"Top down" approach begin with the whole set and proceed to divide it into successively smaller clusters.

Types of Clustering



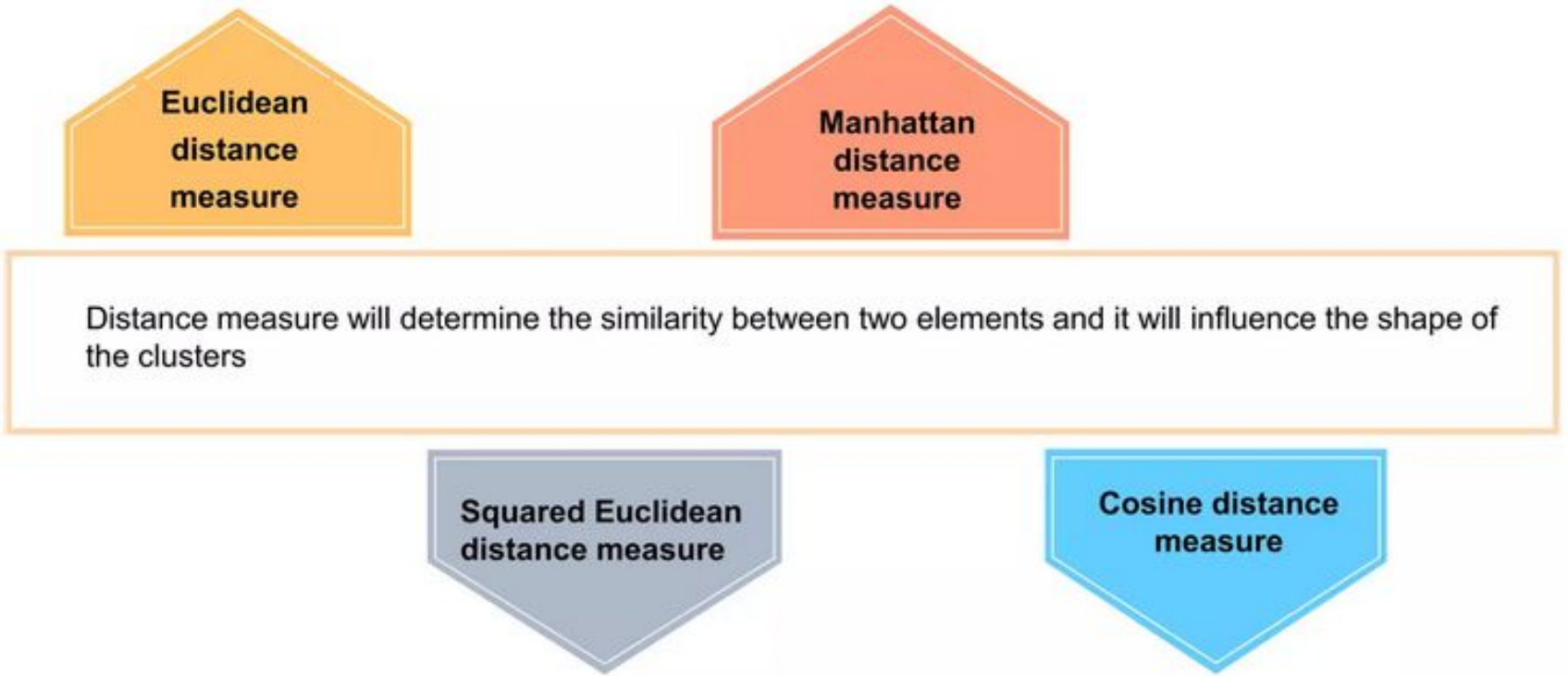
Types of Clustering





Division of objects into clusters such that each object is in exactly one cluster, not several

Distance Measure



Euclidean Distance Measure

01 **Euclidean distance measure**

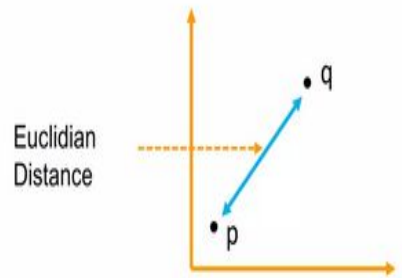
02 Squared euclidean distance measure

03 Manhattan distance measure

04 Cosine distance measure

- The Euclidean distance is the "ordinary" straight line
- It is the distance between two points in Euclidean space

$$d=\sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}$$



Squared Euclidean Distance Measure

01 **Euclidean distance measure**

02 Squared euclidean distance measure

03 Manhattan distance measure

04 Cosine distance measure

The **Euclidean squared distance** metric uses the same equation as the **Euclidean distance** metric, but does not take the square root.

$$d=\sum_{i=1}^{n}(q_i-p_i)^2$$

Manhattan Distance Measure

01 Euclidean distance measure

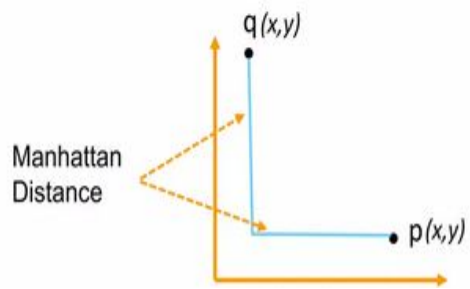
02 Squared euclidean distance measure

03 Manhattan distance measure

04 Cosine distance measure

The Manhattan distance is the simple sum of the horizontal and vertical components or the distance between two points measured along axes at right angles

$$d = \sum_{i=1}^{n} | q_x - p_x | + | q_x - p_y |$$



Cosine Distance Measure

01 Euclidean distance measure

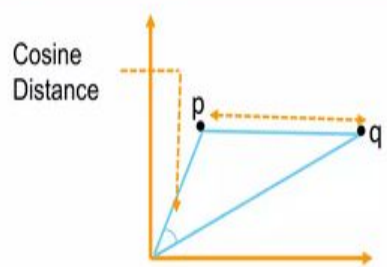
02 Squared euclidean distance measure

03 Manhattan distance measure

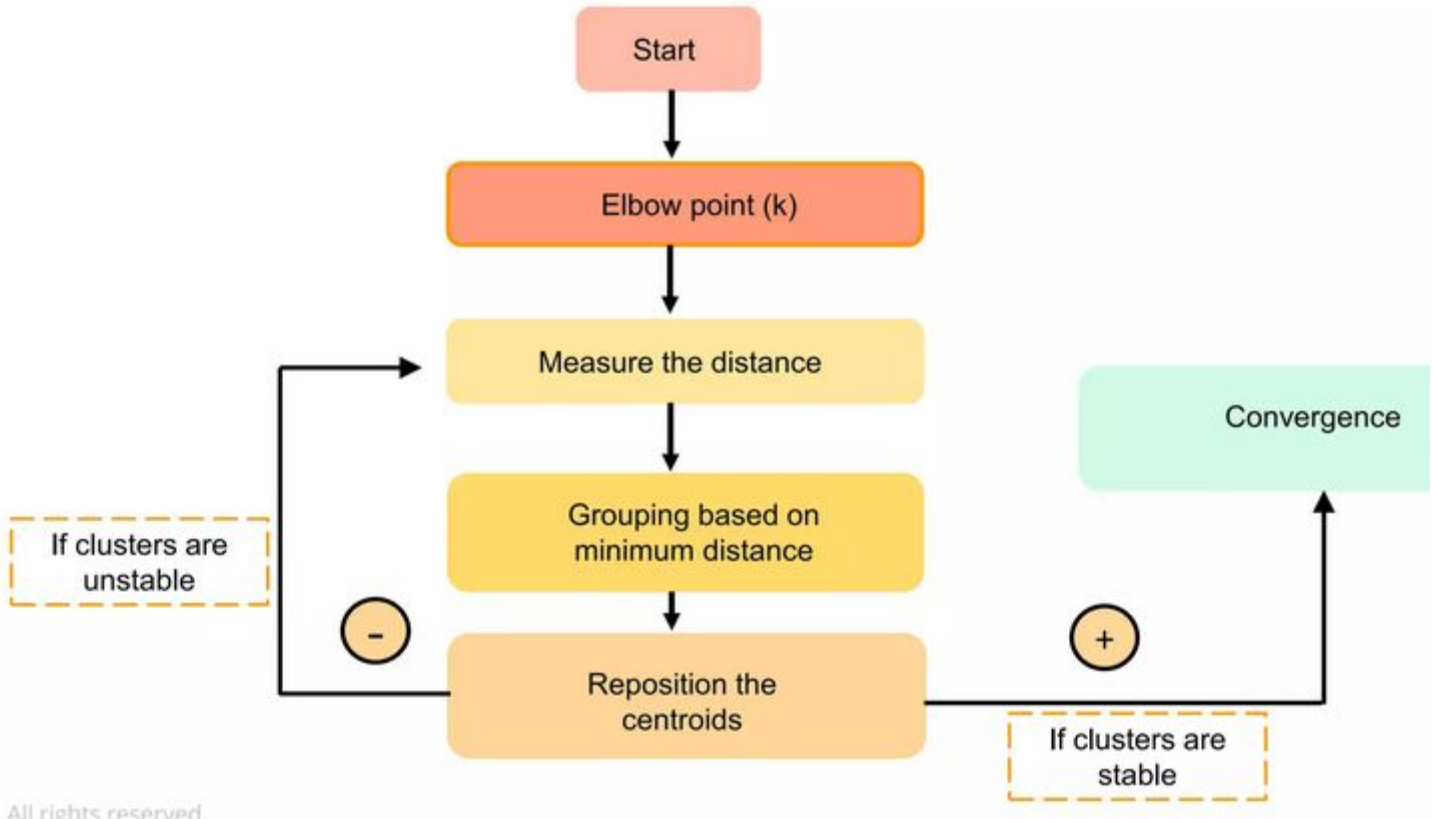
04 Cosine distance measure

The cosine distance similarity measures the angle between the two vectors

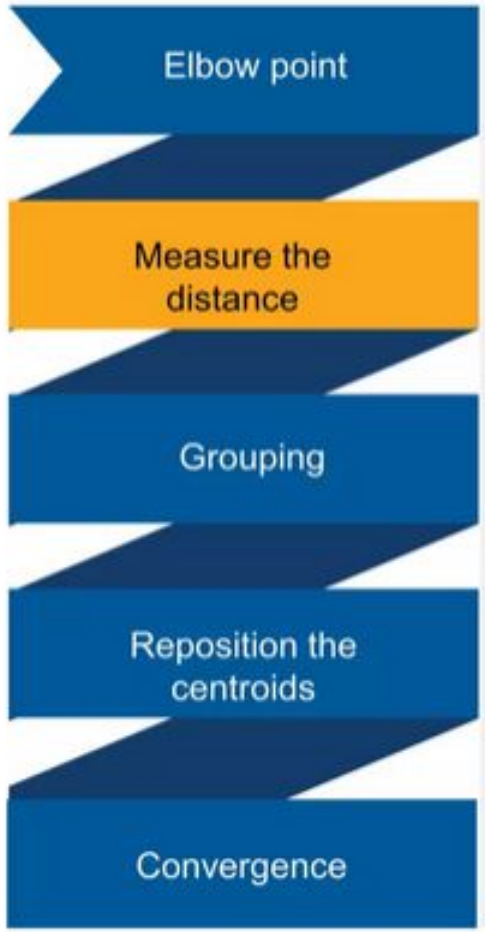
$$d = \frac{\sum_{i=0}^{n-1} q_i - p_x}{\sum_{i=0}^{n-1} (q_i)^2 \times \sum_{i=0}^{n-1} (p_i)^2}$$



How does K-Means clustering work?



How does K-Means clustering work?



Step 1: The given data points below are assumed as **delivery points**

How does K-Means clustering work?

- Elbow point
- Measure the distance
- Grouping
- Reposition the centroids
- Convergence

Step 2: We can randomly initialize two points called the cluster centroids,

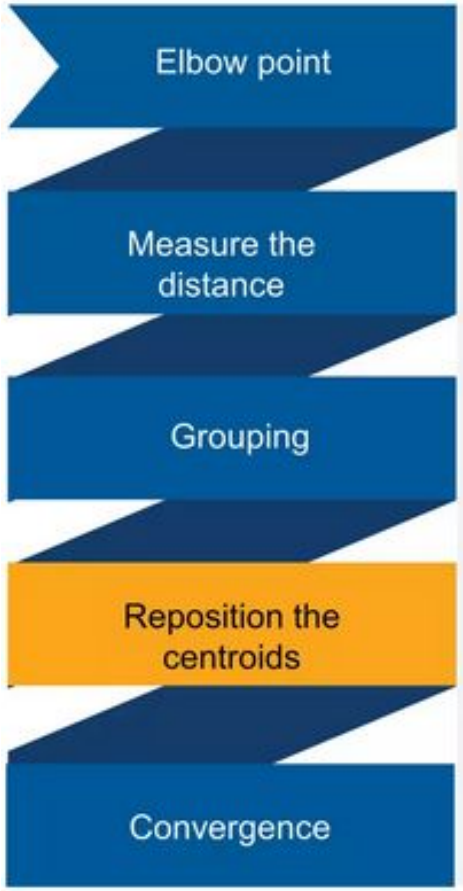
Euclidean distance is a distance measure used to find out which data point is closest to our centroids

How does K-Means clustering work?

- Elbow point
- Measure the distance
- Grouping
- Reposition the centroids
- Convergence

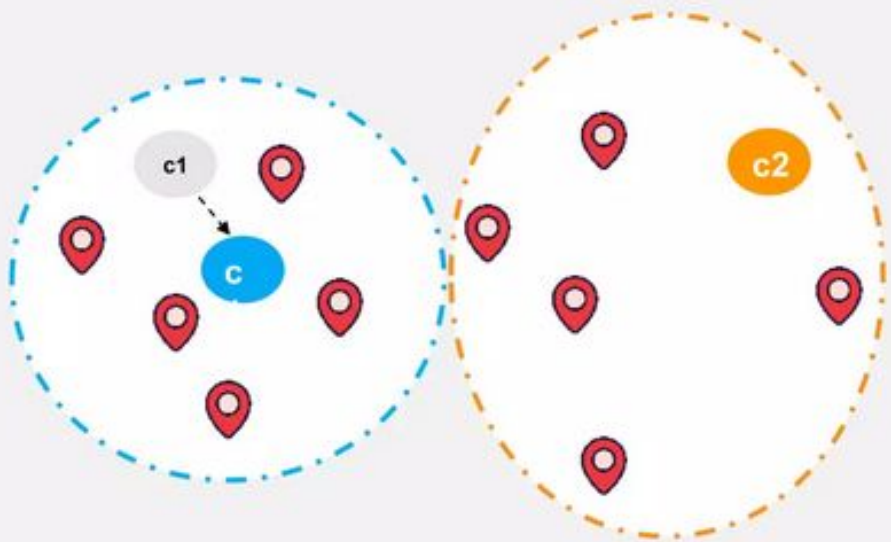
Step 3: Based upon the distance from c1 and c2 centroids, the data points will group itself into clusters

How does K-Means clustering work?



Step 4: Compute the centroid of data points inside blue cluster

Step 5: Reposition the centroid of the blue cluster to the new centroid



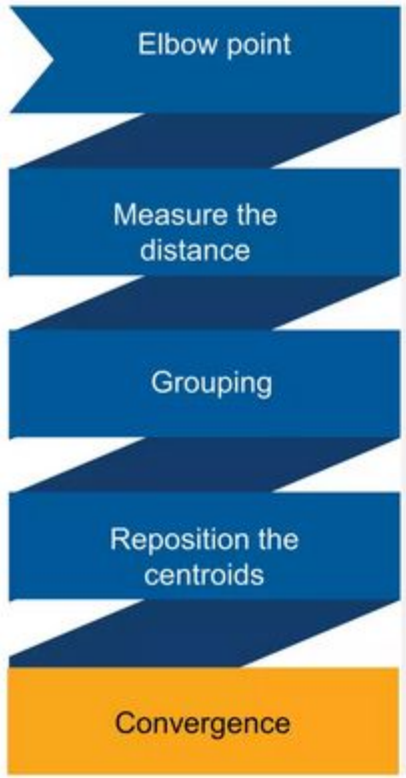
How does K-Means clustering work?

- Elbow point
- Measure the distance
- Grouping
- Reposition the centroids
- Convergence

Step 6: Now, compute the centroid of data points inside orange cluster

Step 7: Reposition the centroid of the orange cluster to the new centroid

How does K-Means clustering work?



Step 8: Once the clusters become static, K-Means clustering algorithm is said to be converged

K-Means Clustering Algorithm

Assuming we have inputs x1,x2,x3,…, and value of **K,**

Step 1 : Pick K random points as cluster centers called centroids

Step 2 : Assign each xi to nearest cluster by calculating its distance to each centroid

Step 3 : Find new cluster center by taking the average of the assigned points

Step 4 : Repeat Step 2 and 3 until none of the cluster assignments change

K-Means Clustering Algorithm

Step 1 :

We randomly pick **K** cluster centers (centroids). Let's assume these are c1,c2,…,ckc1,c2,…,ck, and we can say that;

$$C = c_1, c_2, \ldots c_k$$

C is the set of all centroids.

Step 2:

In this step, we assign each data point to closest center, this is done by calculating Euclidean distance

$$\arg\min_{c_i \in C} dist(\ , x_i)^2$$

Where **dist()** is the Euclidean distance.

K-Means Clustering Algorithm

Step 3:

In this step, we find the new centroid by taking the average of all the points assigned to that cluster.

$$c_i = \frac{1}{|Si|} \sum_{xi \in Si} x_i$$

S_i is the set of all points assigned to the i th cluster

Step 4:

In this step, we repeat step 2 and 3 until none of the cluster assignments change

That means until our clusters remain stable, we repeat the algorithm

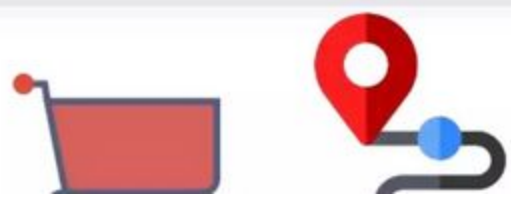
Demo: K-Means Clustering

Problem Statement

- Walmart wants to open a chain of stores across Florida and wants to find out optimal store locations to maximize revenue

Solution

- Walmart already has a strong e-commerce presence
- Walmart can use its online customer data to analyze the customer locations along with the monthly sales

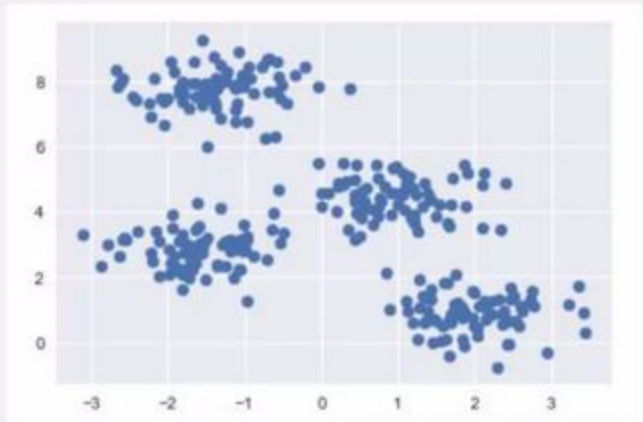


Demo: K-Means Clustering

```
%matplotlib inline
import matplotlib.pyplot as plt

# for plot styling
import seaborn as sns; sns.set()
import numpy as np
from sklearn.datasets.samples_generator import make_blobs
X, y_true = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)
plt.scatter(X[:, 0], X[:, 1], s=50);
```

Demo: K-Means Clustering

```
# output
```



Demo: K-Means Clustering

```
# assign four clusters
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)

# import library
from sklearn.metrics import pairwise_distances_argmin
def find_clusters(X, n_clusters, rseed=2):

# 1. randomly choose clusters
    rng = np.random.RandomState(rseed)
    i = rng.permutation(X.shape[0])[:n_clusters]
    centers = X[i]

    while True:
```

Demo: K-Means Clustering

```
# 2. assign labels based on closest center
     labels = pairwise_distances_argmin(X, centers)

# 3. find new centers from means of points
        new_centers = np.array([X[labels == i].mean(0)
                                for i in range(n_clusters)])

centers, labels = find_clusters(X, 4)
plt.scatter(X[:, 0], X[:, 1], c=labels,
            s=50, cmap='viridis')
```

Demo: K-Means Clustering

```
# 4. check for convergence
   if np.all(centers == new_centers):
            break
   centers = new_centers
return centers, labels
centers, labels = find_clusters(X, 4)
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')

plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5);
```

Demo: K-Means Clustering

output:

Conclusion

Congratulations!

We have demonstrated K-Means clustering by establishing Walmart stores across Florida in the most optimized way

Use Case: K-Means for Color Compression

Problem Statement

To perform color compression on images using K-Means algorithm

Use Case: K-Means for Color Compression

```
# example 1:

# note: this requires the ``pillow`` package to be installed
from sklearn.datasets import load_sample_image
china = load_sample_image("flower.jpg")
ax = plt.axes(xticks=[], yticks=[])
ax.imshow(china);
```

#Output:

Use Case: K-Means for Color Compression

```
# returns the dimensions of the array
china.shape
```

```
Out[26]: (427, 640, 3)
```

```
# reshape the data to [n_samples x n_features], and rescale the colors so that they lie between 0 and 1
data = china / 255.0 # use 0...1 scale
data = data.reshape(427 * 640, 3)
data.shape
```

```
Out[27]: (273280, 3)
```

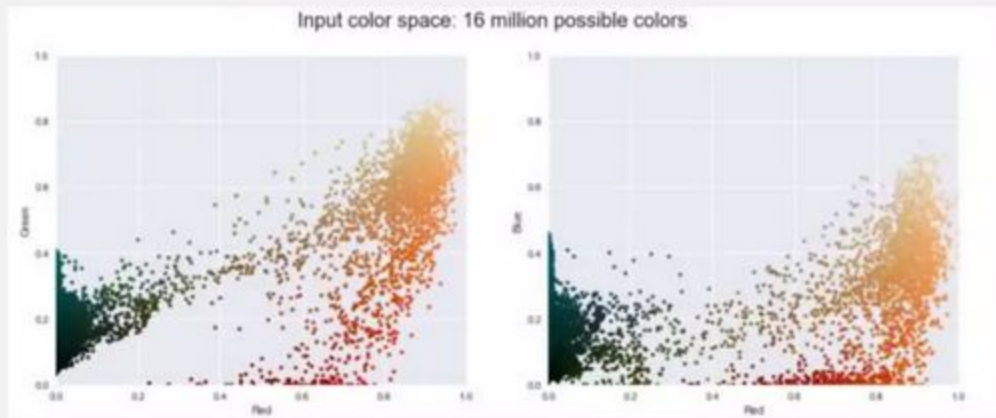
```
# visualize these pixels in this color space, using a subset of 10,000 pixels for efficiency
def plot_pixels(data, title, colors=None, N=10000):
    if colors is None:
        colors = data
```

Use Case: K-Means for Color Compression

```
# choose a random subset
    rng = np.random.RandomState(0)
    i = rng.permutation(data.shape[0])[:N]
    colors = colors[i]
    R, G, B = data[i].T

    fig, ax = plt.subplots(1, 2, figsize=(16, 6))
    ax[0].scatter(R, G, color=colors, marker='.')
    ax[0].set(xlabel='Red', ylabel='Green', xlim=(0, 1), ylim=(0, 1))

    ax[1].scatter(R, B, color=colors, marker='.')
    ax[1].set(xlabel='Red', ylabel='Blue', xlim=(0, 1), ylim=(0, 1))

    fig.suptitle(title, size=20);
```

Use Case: K-Means for Color Compression

```
plot_pixels(data, title='Input color space: 16 million possible colors')
```



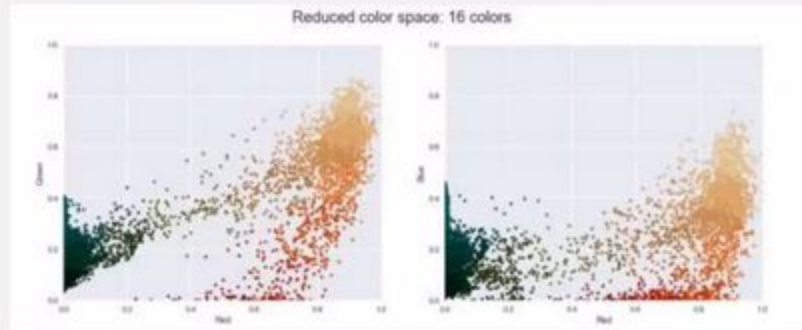
Use Case: K-Means for Color Compression

```
# fix numPy issues
import warnings; warnings.simplefilter('ignore’)

# reducing these 16 million colors to just 16 colors
from sklearn.cluster import MiniBatchKMeans
kmeans = MiniBatchKMeans(16)
kmeans.fit(data)
new_colors = kmeans.cluster_centers_[kmeans.predict(data)]

plot_pixels(data, colors=new_colors,
            title="Reduced color space: 16 colors")
```

Use Case: K-Means for Color Compression

the result is re-coloring of the original pixels, where each pixel is assigned the color of its closest cluster center

output:




```
china_recolored = new_colors.reshape(china.shape)
fig, ax = plt.subplots(1, 2, figsize=(16, 6), subplot_kw=dict(xticks=[], yticks=[]))
fig.subplots_adjust(wspace=0.05)
ax[0].imshow(china)
ax[0].set_title('Original Image', size=16)
ax[1].imshow(china_recolored)
ax[1].set_title('16-color Image', size=16);
```

Use Case: K-Means for Color Compression

output

Use Case: K-Means for Color Compression

```
# example 2:

from sklearn.datasets import load_sample_image
china = load_sample_image("china.jpg")
ax = plt.axes(xticks=[], yticks=[])
ax.imshow(china);
```

Use Case: K-Means for Color Compression

```
# returns the dimensions of the array
china.shape
```

```
Out[26]: (427, 640, 3)
```

```
# reshape the data to [n_samples x n_features], and rescale the colors so that they lie between 0 and 1
data = china / 255.0 # use 0...1 scale
data = data.reshape(427 * 640, 3)
data.shape
```

```
Out[27]: (273280, 3)
```

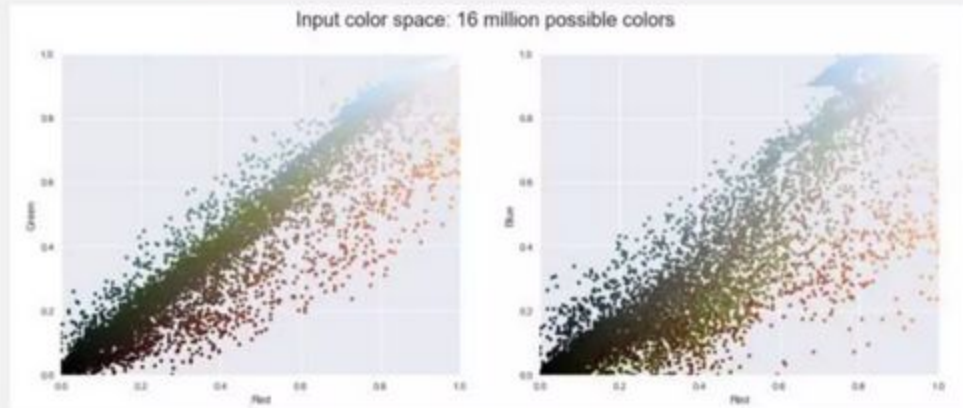
```
# visualize these pixels in this color space, using a subset of 10,000 pixels for efficiency
def plot_pixels(data, title, colors=None, N=10000):
    if colors is None:
        colors = data
```

Use Case: K-Means for Color Compression

```
# choose a random subset
    rng = np.random.RandomState(0)
    i = rng.permutation(data.shape[0])[:N]
    colors = colors[i]
    R, G, B = data[i].T

    fig, ax = plt.subplots(1, 2, figsize=(16, 6))
    ax[0].scatter(R, G, color=colors, marker='.')
    ax[0].set(xlabel='Red', ylabel='Green', xlim=(0, 1), ylim=(0, 1))

    ax[1].scatter(R, B, color=colors, marker='.')
    ax[1].set(xlabel='Red', ylabel='Blue', xlim=(0, 1), ylim=(0, 1))

    fig.suptitle(title, size=20);
```

Use Case: K-Means for Color Compression

```
plot_pixels(data, title='Input color space: 16 million possible colors')
```

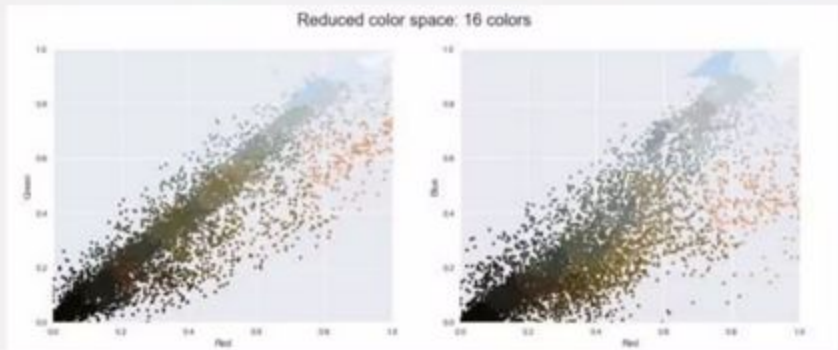


Use Case: K-Means for Color Compression

```
# fix NumPy issues
import warnings; warnings.simplefilter('ignore')

# reducing these 16 million colors to just 16 colors
from sklearn.cluster import MiniBatchKMeans
kmeans = MiniBatchKMeans(16)
kmeans.fit(data)
new_colors = kmeans.cluster_centers_[kmeans.predict(data)]

plot_pixels(data, colors=new_colors,
            title="Reduced color space: 16 colors")
```

Use Case: K-Means for Color Compression

```
# the result is a re-coloring of the original pixels, where each pixel is assigned the color of its closest cluster center
# output
```




```
china_recolored = new_colors.reshape(china.shape)
fig, ax = plt.subplots(1, 2, figsize=(16, 6), subplot_kw=dict(xticks=[], yticks=[]))
fig.subplots_adjust(wspace=0.05)
ax[0].imshow(china)
ax[0].set_title('Original Image', size=16)
ax[1].imshow(china_recolored)
ax[1].set_title('16-color Image', size=16);
```

Use Case: K-Means for Color Compression

```
# output
```

Conclusion

Congratulations!

We have demonstrated K-Means in color compression.

The hands on example will help you to encounter any K-Means project in future.

Worked out Example

- ***Input:*** Number of Objects = 6, Number of Clusters = 2

No	X	Y
1	1	1
2	2	3
3	1	2
4	3	3
5	2	2
6	3	1

- Step-1: Choose random K points and set as cluster centers

C1 = (2, 2) C2 = (3 , 3)

- Step-2: Calculating the distance between objects into cluster centroids by using Euclidean Distance

$$D_i = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

1. D1 $= \{(1, 1), (2, 2)\}$

$= \sqrt{(2-1)^2 + (2-1)^2}$

$= 1.41$

2. D1 $= \{(2, 3), (2, 2)\}$

$= \sqrt{(2-2)^2 + (2-3)^2}$

$= 1$

3. D1 $= \{(1, 2), (2, 2)\}$

$= \sqrt{(2-1)^2 + (2-2)^2}$

1. D2 $= \{(1, 1), (3, 3)\}$

$= \sqrt{(3-1)^2 + (3-1)^2}$

$= 2.82$

2. D2 $= \{(2, 3), (3, 3)\}$

$= \sqrt{(3-2)^2 + (3-3)^2}$

$= 1$

3. D2 $= \{(1, 2), (3, 3)\}$

$= \sqrt{(3-1)^2 + (3-2)^2}$

4. D1 $= \{(3 , 3), (2 , 2)\}$

$= \sqrt{(2-3)^2 + (2-3)^2}$

$= 1.41$

5. D1 $= \{(2 , 2), (2 , 2)\}$

$= \sqrt{(2-2)^2 + (2-2)^2}$

$= 0$

6. D1 $= \{(3 , 1), (2 , 2)\}$

$= \sqrt{(2-3)^2 + (2-1)^2}$

$= 1.41$

4. D2 $= \{(3 , 3), (3 , 3)\}$

$= \sqrt{(3-3)^2 + (3-3)^2}$

$= 0$

5. D2 $= \{(2 , 2), (3 , 3)\}$

$= \sqrt{(3-2)^2 + (3-2)^2}$

$= 1.41$

6. D2 $= \{(3 , 1), (3 , 3)\}$

$= \sqrt{(3-3)^2 + (3-1)^2}$

$= 2$

C1 = {(1 , 1), (1 , 2), (2 , 2), (3 , 1)}

C2 = {(2 , 3), (3 , 3)}

- Step-3: Recalculating the position of the centroid

$$\text{Mean} = \left(\frac{x_1+x_2+\ldots.x_n}{n}, \frac{y_1+y_2+\ldots.y_n}{n}\right)$$

$$C1 = \left(\frac{1+1+2+3}{4}, \frac{1+2+2+1}{4}\right)$$

New $C1 = (1.75, 1.5)$

$$C2 = \left(\frac{2+3}{2}, \frac{3+3}{2}\right)$$

New $C2 = (2.5, 3)$

No	X	Y
1	1	1
2	2	3
3	1	2
4	3	3
5	2	2
6	3	1

- Step-4: Go back to Step 2, unless the centroids are not changing.

1. For the given data, compute two clusters using K-means algorithm for clustering where initial cluster centers are (1.0, 1.0) and (5.0, 7.0). Execute for two iterations.

Record Number	A	B
R1	1.0	1.0
R2	1.5	2.0
R3	3.0	4.0
R4	5.0	7.0
R5	3.5	5.0
R6	4.5	5.0
R7	3.5	4.5

Initialization: Number of clusters (K) = 2, centroid for cluster1 (C1)= (1.0, 1.0) and centroid for cluster2 (C2) = (5.0, 7.0). We use Euclidean distance to find closest point to centroids.

Iteration1:

Record Number	**Close to C1**(1.0, 1.0)	**Close to C2**(5.0, 7.0)	**Assign to cluster**
R1(1.0,1.0)	dist(R1, C1)=0.0	dist(R1, C2)=7.21	Cluster1
R2(1.5,2.0)	dist(R2, C1)=1.12	dist(R2, C2)=6.12	Cluster1
R3(3.0,4.0)	dist(R3, C1)=3.61	dist(R3, C2)=3.61	Cluster1
R4(5.0,7.0)	dist(R4, C1)=7.21	dist(R4, C2)=0.0	Cluster2
R5(3.5,5.0)	dist(R5, C1)=4.12	dist(R5, C2)=2.5	Cluster2
R6(4.5,5.0)	dist(R6, C1)= 5.31	dist(R6, C2)=2.06	Cluster2
R7(3.5,4.5)	dist(R7,C1)=4.30	dist(R7, C2)=2.92	Cluster2

Thus, we obtain two clusters containing:

Cluster1 {R1, R2, R3} and Cluster2 {R4, R5, R6, R7}.

Their new centroids are:

C1 = (1.0+1.5+3.0)/3, (1.0+2.0+4.0)/3
= 5.5/3, 7.0/3
= 1.83, 2.33

C2 = (5.0+3.5+4.5+3.5)/4, (7+5+5+4.5)/4
= 16.5/4, 21.5/4
= 4.12, 5.37

Iteration2:

Record Number	**Close to C1**(1.83, 2.33)	**Close to C2**(4.12, 5.37)	**Assign to cluster**
R1(1.0,1.0)	dist(R1, C1)=1.57	dist(R1, C2)=5.37	Cluster1
R2(1.5,2.0)	dist(R2, C1)=0.47	dist(R2, C2)=4.27	Cluster1
R3(3.0,4.0)	dist(R3, C1)=2.04	dist(R3, C2)=1.77	Cluster2
R4(5.0,7.0)	dist(R4, C1)=5.64	dist(R4, C2)=1.85	Cluster2
R5(3.5,5.0)	dist(R5, C1)=3.15	dist(R5, C2)=0.72	Cluster2
R6(4.5,5.0)	dist(R6, C1)=3.78	dist(R6, C2)=0.53	Cluster2
R7(3.5,4.5)	dist(R7,C1)=2.74	dist(R7, C2)=1.07	Cluster2

Therefore, new clusters are:

Cluster1 {R1, R2} and Cluster2 {R3, R4, R5, R6, R7}.

Their new centroids are:

C1 = (1.0+1.5)/2, (1.0+2.0)/2
= 2.50/2, 3.0/2
= 1.25, 1.5

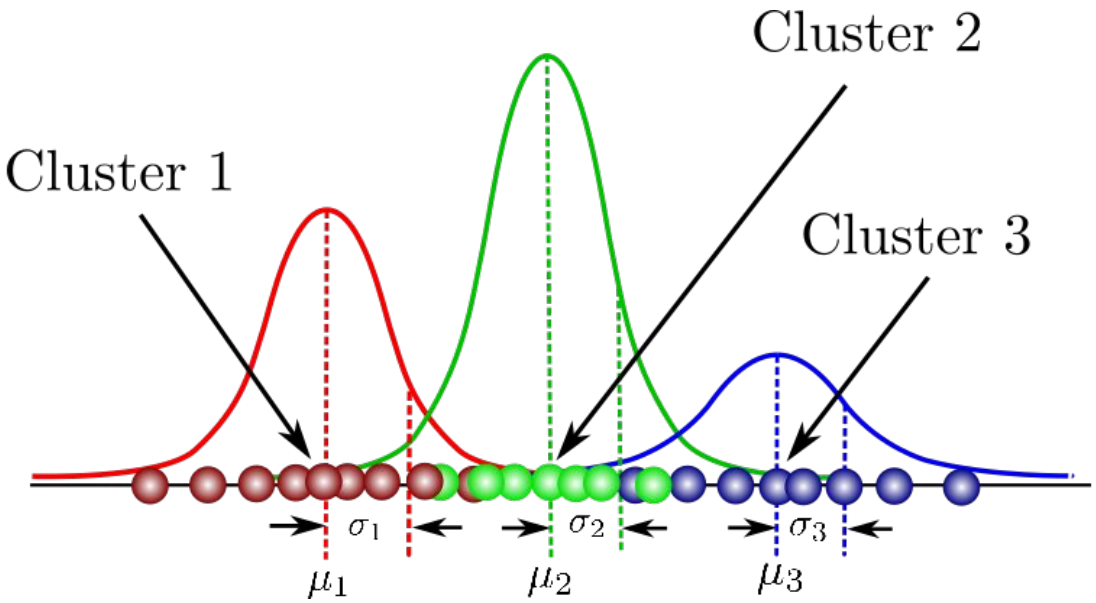
C2 = (3.0+5.0+3.5+4.5+3.5)/5, (4+7+5+5+4.5)/5
= 19.5/5, 25.5/5
= 3.9, 5.1

Gaussian Mixture Model (GMM) algorithm

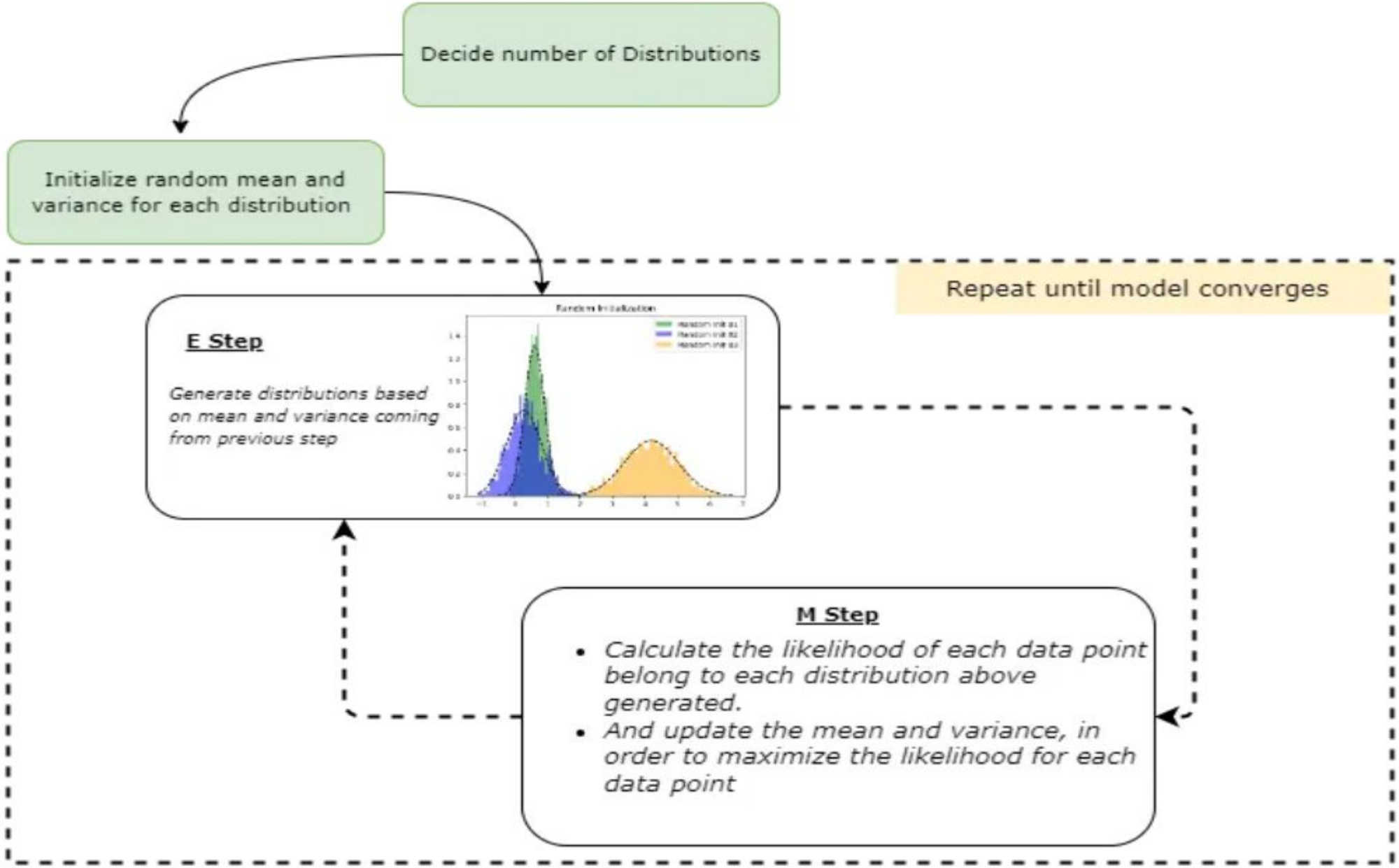
K-Means uses a distance-based approach, and GMM uses a probabilistic approach.

There is one primary assumption in GMM:

The dataset consists of multiple Gaussians, in other words, a mixture of the Gaussian.



Peak represents the different Gaussian distribution or the cluster in our dataset.



Summary of EM steps of GMM | Image by Author

Many of the datasets can be easily modeled with the help of Gaussian Distribution. Therefore, one can assume that the clusters from different Gaussian Distributions.

The single dimension probability density function of a Gaussian Distribution is as follows –

$$y = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$\mu =$ Mean
$\sigma =$ Standard Deviation
$\pi \approx 3.14159\cdots$
$e \approx 2.71828\cdots$

two types of values

Gaussian Mixture Model – 1. component weights 2.variances/covariances

A Gaussian Mixture Model with K components, $\boldsymbol{\mu_k}$ is the mean of the k^{th} component.

univariate case : will have a variance of $\boldsymbol{\sigma_k}$

multivariate case : will have a covariance matrix of $\boldsymbol{\Sigma_k}$.

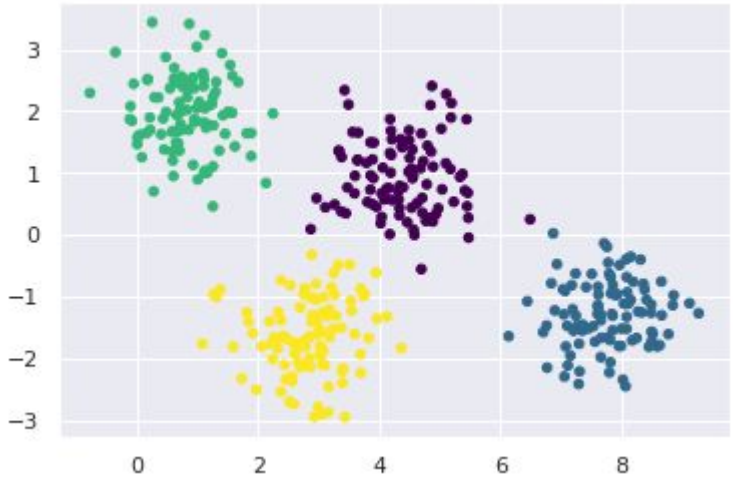
$\boldsymbol{\Phi_k}$ is the definition of the mixture component weights which is for every component $\boldsymbol{C_k}$.

This has a constraint that $\sum_{i=1}^{K} \boldsymbol{\phi}_i = 1$ such that the total probability gets normalized to 1.

$$p(x) = \sum_{i=1}^{K} \phi_i \mathcal{N}(x \mid \mu_i, \sigma_i)$$

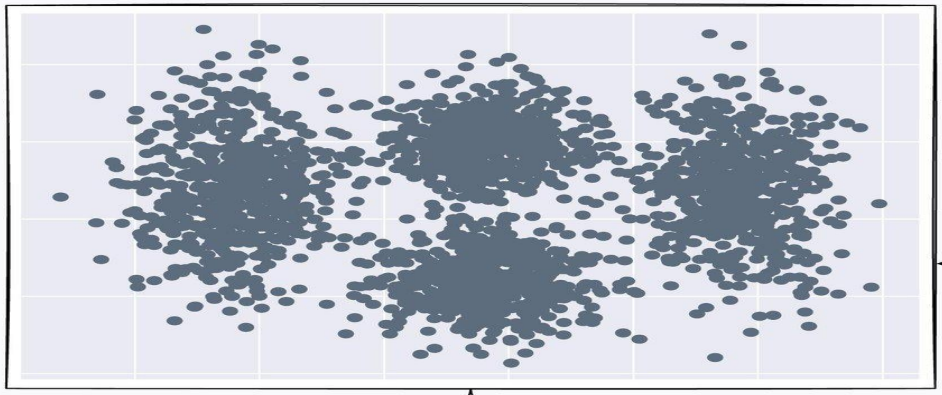
$$\mathcal{N}(x \mid \mu_i, \sigma_i) = \frac{1}{\sigma_i \sqrt{2\pi}} \exp\left(-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right)$$

$$\sum_{i=1}^{K} \phi_i = 1$$



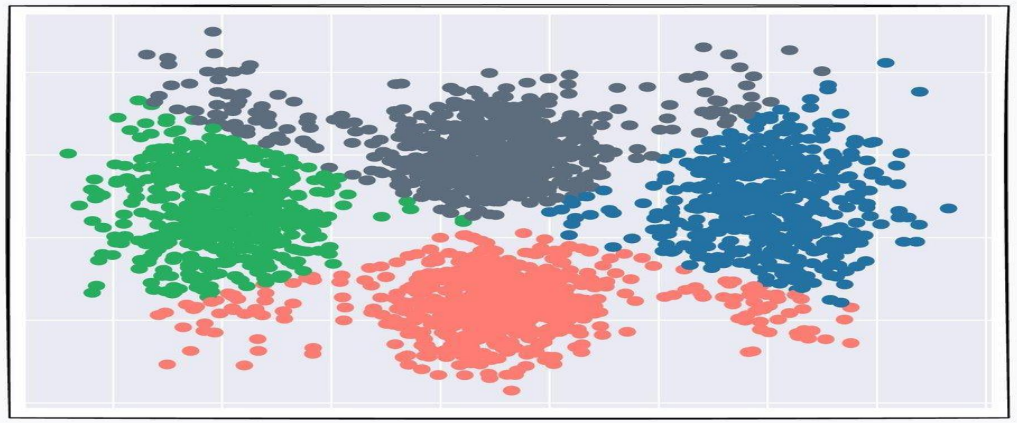
Gaussian Mixture Model vs. KMeans

blog.DailyDoseofDS.com



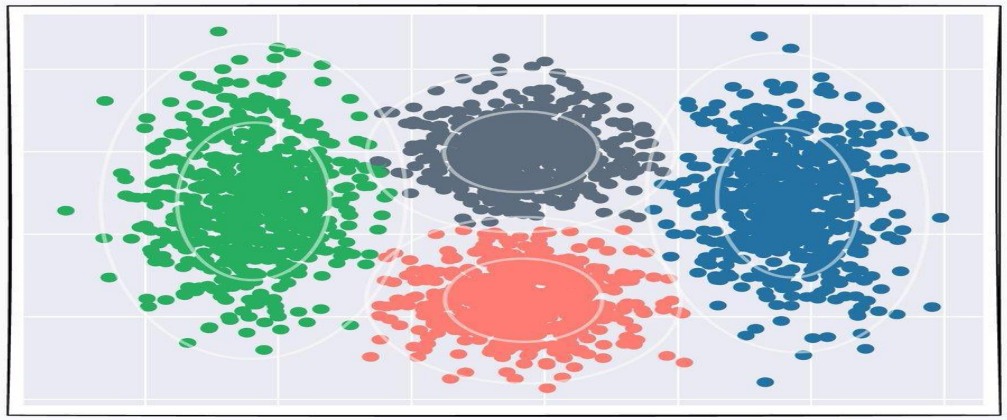
Clusters with different variances

KMeans



Incorrect clustering

Gaussian Mixture Model

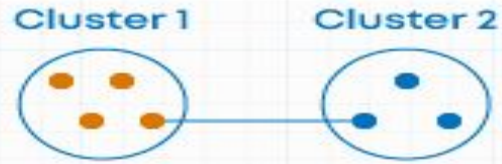


Correct clustering

- **Single Linkage**

  $D(c_1,c_2) = \min D(x_1,x_2)$

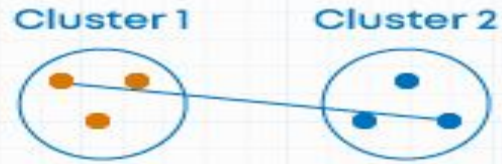
  Minimum distance or distance between closest elements in clusters



- **Complete Linkage**

  $D(c_1,c_2) = \max D(x_1,x_2)$

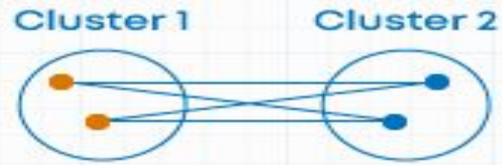
  Maximum distance between elements in clusters



- **Average Linkage**

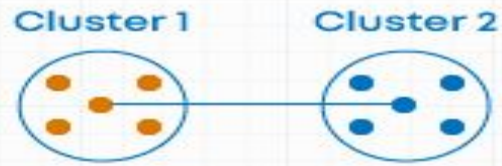
  $D(c_1,c_2) = \frac{1}{|c_1|}\frac{1}{|c_2|}\Sigma\Sigma D(x_1,x_2)$

  Average of the distances of all pairs

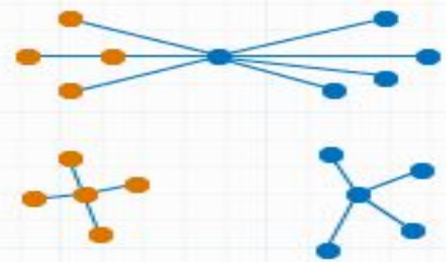


- **Centroid Method**

  Combining clusters with minimum distance between the centroids of the two clusters



- **Ward's Method**
  - Combining clusters where increase in within cluster variance is to the smallest degree.
  - Objective is to minimize the total within cluster vairance



Instance-Based Learning

Idea:

- Similar examples have similar label.
- Classify new examples like similar training examples.

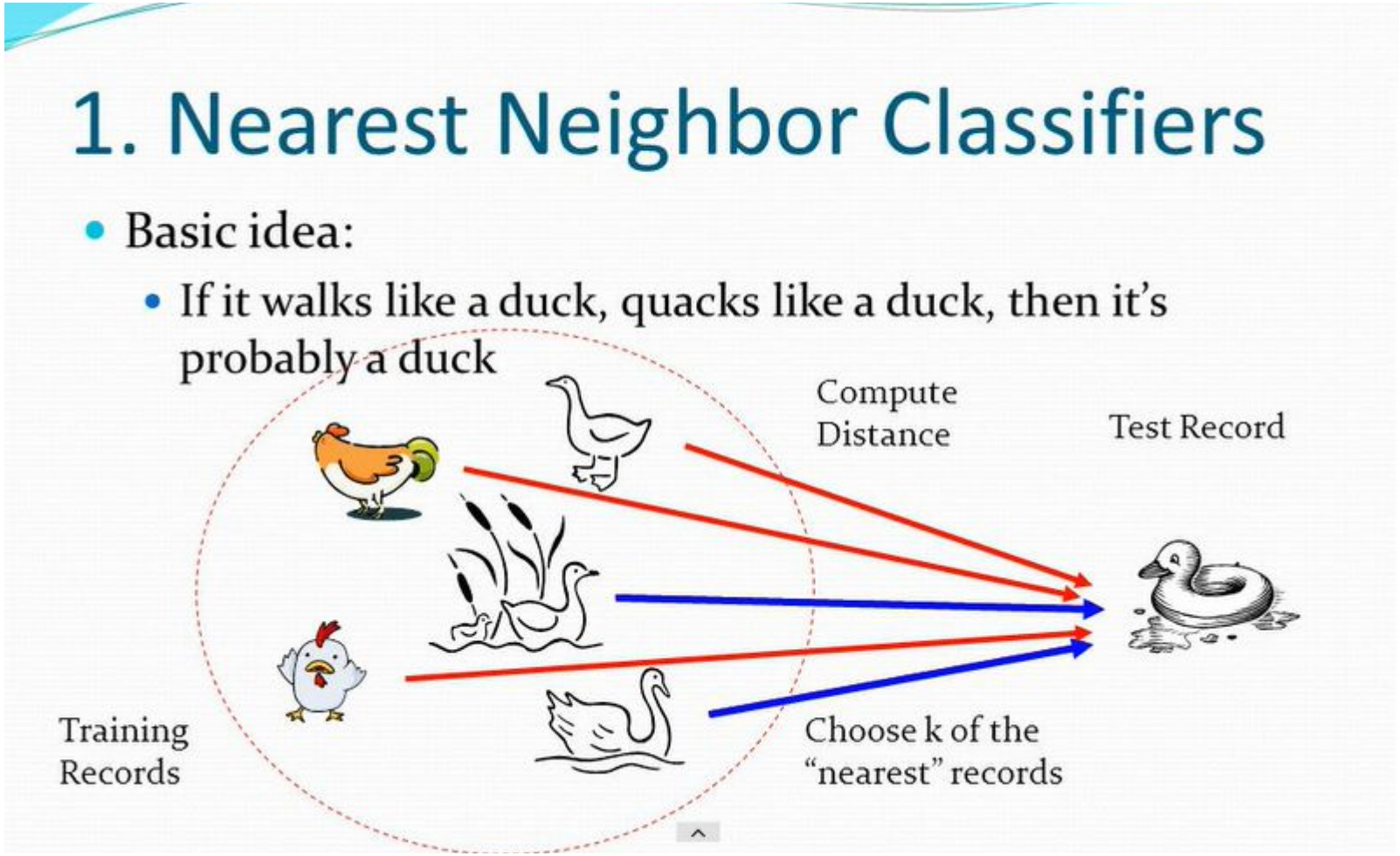
Algorithm:

- Given some new example x for which we need to predict its class y
- Find most similar training examples
- Classify *x* "like" these most similar examples

Questions:

- How to determine similarity?
- How many similar training examples to consider?
- How to resolve inconsistencies among the training examples?

The most basic instance-based method is the k-NEAREST NEIGHBOR algorithm



What is KNN?

KNN is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry.

- **Lazy learning algorithm**
- **Non-parametric learning algorithm**

1. **Lazy learning algorithm** – KNN is a lazy learning algorithm because it does not have a specialized training phase or model and uses all the data for training while classification.

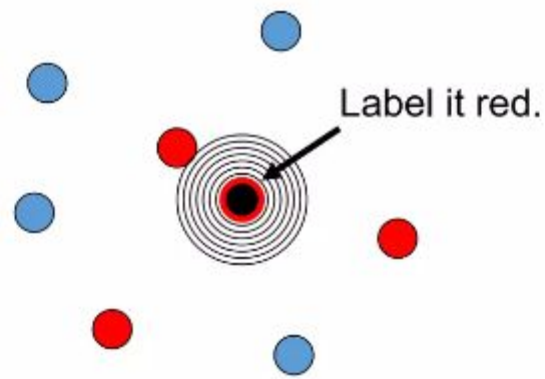
2. **Non-parametric learning algorithm** – KNN is also a non-parametric learning algorithm because it doesn't assume anything about the underlying data.

3. **Eager learning algorithm** - Eager learners, when given a set of training tuples, will construct a generalization model before receiving new (e.g., test) tuples to classify.

Nearest Neighbor

One of the simplest of all machine learning classifiers

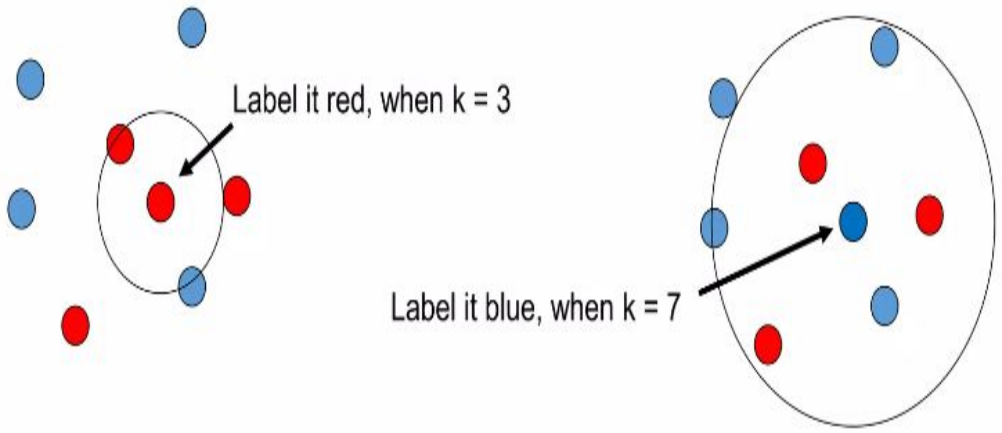
Simple idea: label a new point the same as the closest known point



K Nearest Neighbor

Generalizes 1-NN to smooth away noise in the labels

A new point is now assigned the most frequent label of its *k* nearest neighbors



Let an arbitrary instance ***x*** be described by the feature vector

$$\langle a_1(x), a_2(x), \ldots a_n(x) \rangle$$

where $a_r(x)$ denotes the value of the rth attribute of instance ***x.***

Then the distance between two instances x_i and x_j is defined to be $d(x_i, x_j)$,

where,

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

the target function may be either discrete-valued or real-valued.

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

TABLE 8.1
The k-NEAREST NEIGHBOR algorithm for approximating a discrete-valued function $f : \Re^n \rightarrow V$.

$\hat{f}(x_q)$ = Returned by this algorithm as its estimate of $f(x_q)$ is just the most common value of **f** among the k training examples nearest to x_q.

$f(x_i)$ where x_i is the training instance nearest to x_i.

The nearest neighbors of an instance are defined in terms of the standard Euclidean distance.

Distance Metrics

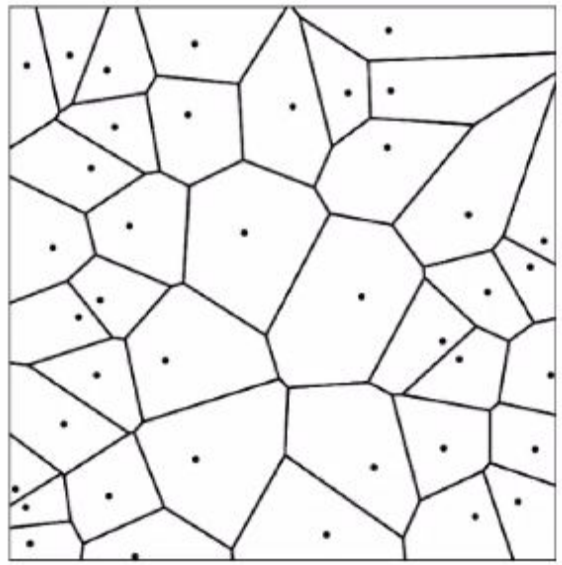
- Euclidean distance
- Manhattan distance
- Hamming distance (for discrete data)
- Others (e.g., normal, cosine)

When different units are used for each dimension normalize each dimension by standard deviation.

1 - Nearest Neighbor

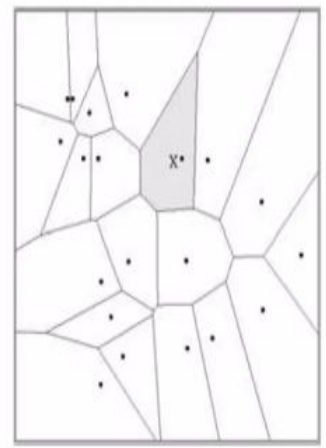
Forms a Voronoi tessellation of the instance space

K = 1

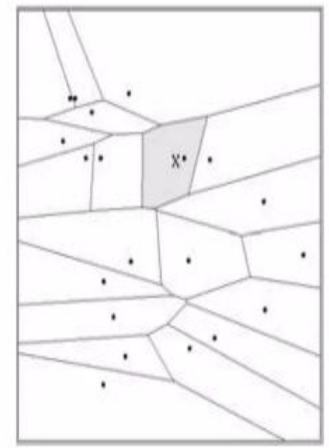


Distance Metrics

Different metrics can change the decision surface



K = 1



$Dist(\mathbf{a},\mathbf{b}) = (a_1 - b_1)^2 + (a_2 - b_2)^2$

$Dist(\mathbf{a},\mathbf{b}) = (a_1 - b_1)^2 + (3a_2 - 3b_2)^2$

Pros and Cons of KNN

Pros

- Simple algorithm to understand and interpret.
- Useful for nonlinear data because there is no assumption about data in this algorithm.
- Versatile algorithm as we can use it for classification as well as regression.
- Has Relatively high accuracy but there are much better supervised learning models than KNN.

Pros and Cons of KNN

Cons

- Computationally a bit expensive algorithm because it stores all the training data.
- High memory storage required as compared to other supervised learning algorithms.
- Prediction is slow in case of big N.
- Sensitive to the scale of data as well as irrelevant features.

Applications of KNN

Banking System

KNN can be used in banking system to predict weather an individual is fit for loan approval? Does that individual have the characteristics similar to the defaulters one?

Calculating Credit Ratings

KNN algorithms can be used to find an individual's credit rating by comparing with the persons having similar traits.

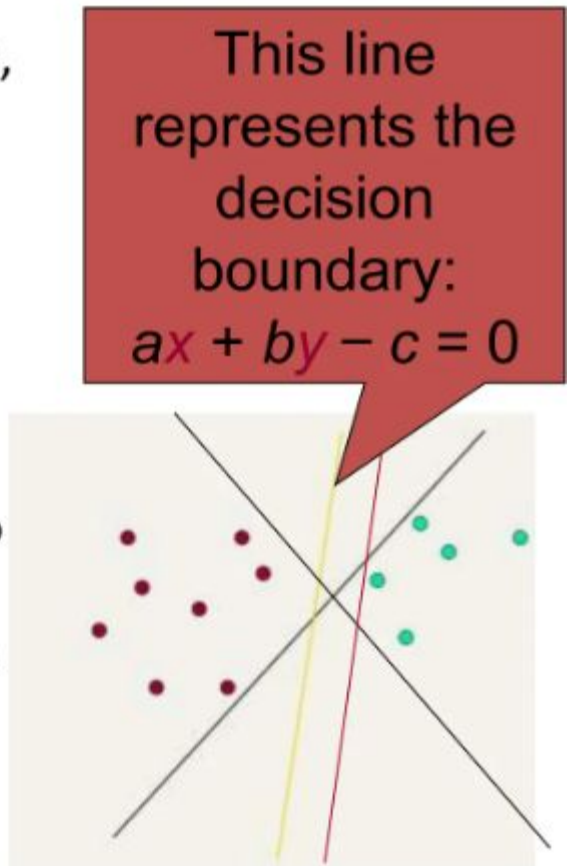
SVM - Support Vector Machines

-Binary Classifier

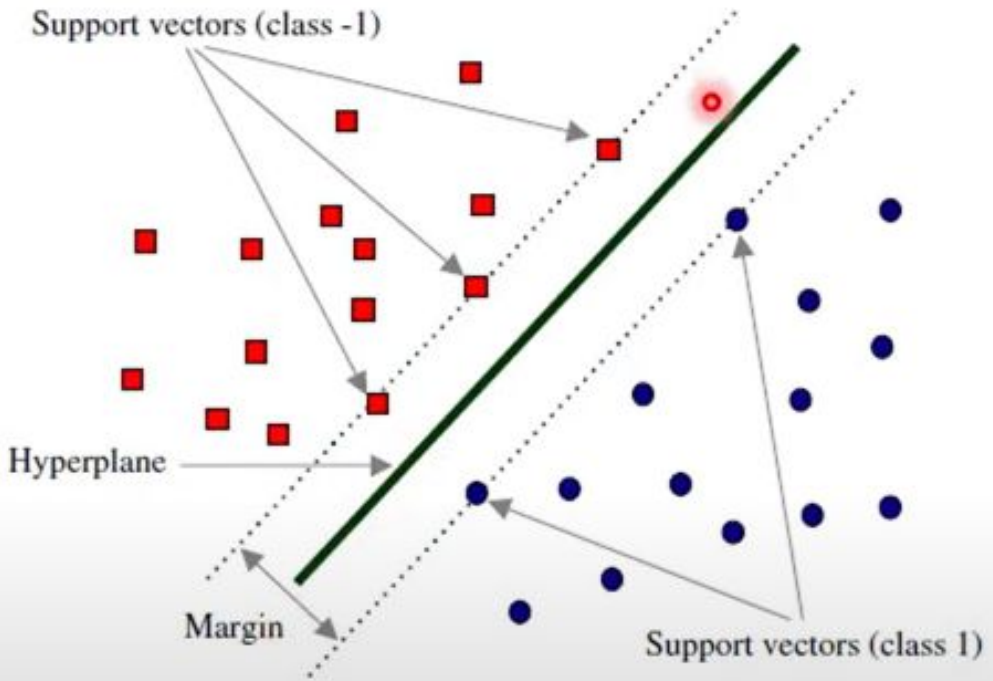
- A new classification method for both linear and nonlinear data
- It uses a nonlinear mapping to transform the original training data into a higher dimension
- With the new dimension, it searches for the linear optimal separating hyperplane (i.e., "decision boundary")
- With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane
- SVM finds this hyperplane using support vectors ("essential" training tuples) and margins (defined by the support vectors)

Linear classifiers: Which Hyperplane?

- Lots of possible choices for *a, b, c.*
- Some methods find a separating hyperplane, but not the optimal one [according to some criterion of expected goodness]
  - E.g., perceptron
- A Support Vector Machine (SVM) finds an optimal* solution.
  - Maximizes the distance between the hyperplane and the "difficult points" close to decision boundary
  - One intuition: if there are no points near the decision surface, then there are no very uncertain classification decisions

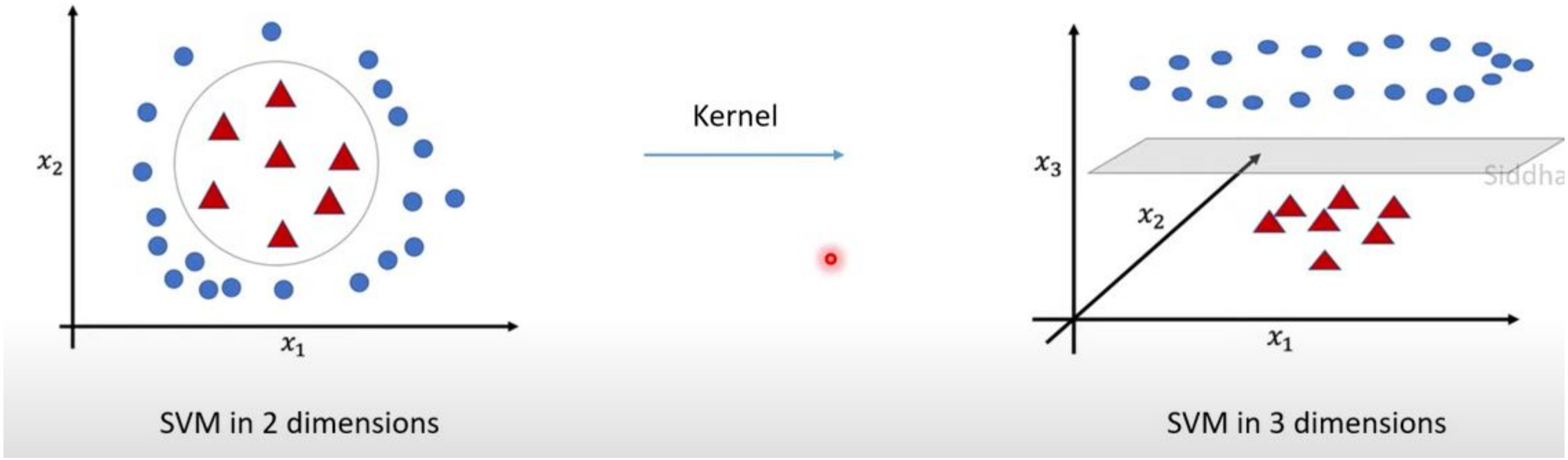


Support Vector Machine Classifier



- Hyperplane
- Support Vectors
- Margin
- Linearly separable data

Non linear Data need more dimensions to classify



Kernel Functions project the low dimensional data to high dimensional space

Kernel is math function …it increases the dimensional

Mathematical Derivation of Hyper plan for SVM Classification

Step 1:

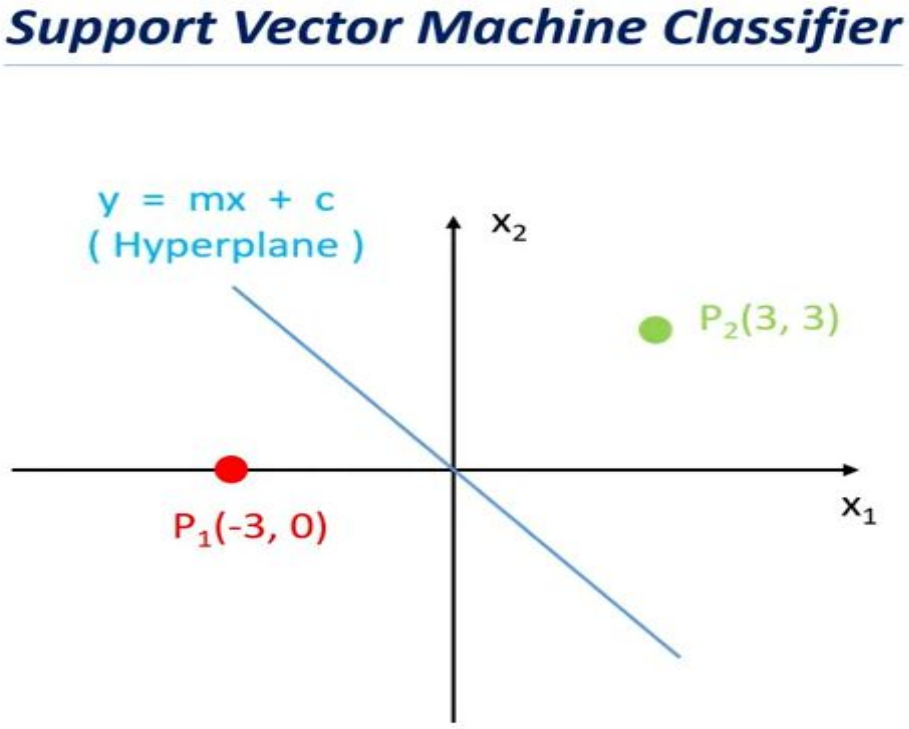
Finding the linear line parameters which passes through the origin (1. slope, 2. intersect)

Step 2:

Projecting the data points on to the line

which is.. □(line parameter)^T * with data points)

Example:



Let slope, m = - 1

Intercept, c = 0

w --> parameters of the line

(m, c) = (-1, 0)

Example:

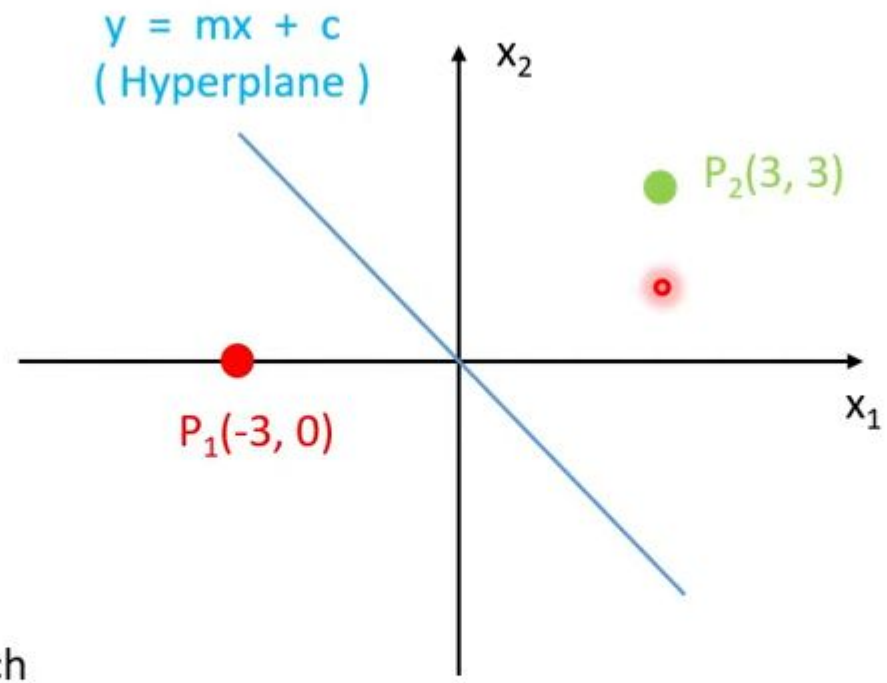
If our slope is positive means the label is vice-versa

$P_1(-3, 0)$

$$w^T x = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \begin{bmatrix} -3 & 0 \end{bmatrix}$$

$w^T x = 3$

(Positive)



$P_2(3, 3)$

$$w^T x = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \begin{bmatrix} 3 & 3 \end{bmatrix}$$

$w^T x = -3$

(Negative)

Inference: For all the points which lie in the left side of the hyperplane, $w^T x$ value will be Positive

Let slope, m = - 1

Intercept, c = 0

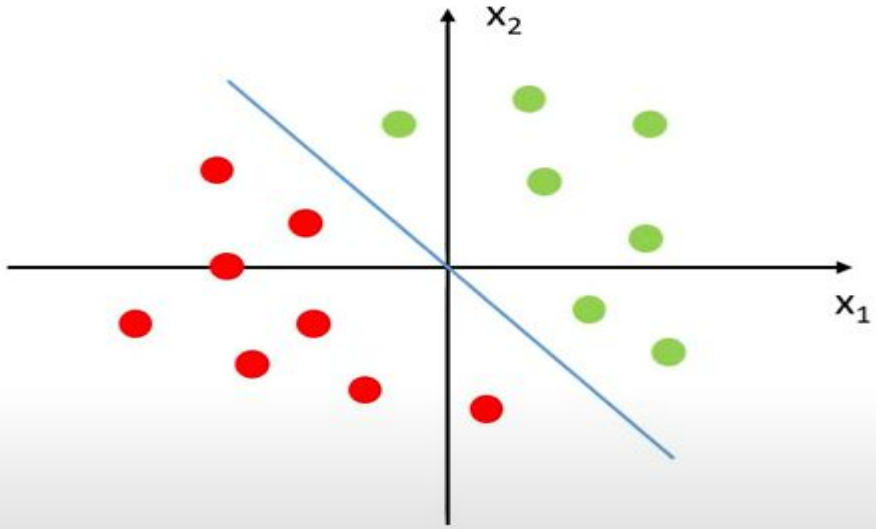
w --> parameters of the line

(m, c) = (-1, 0)

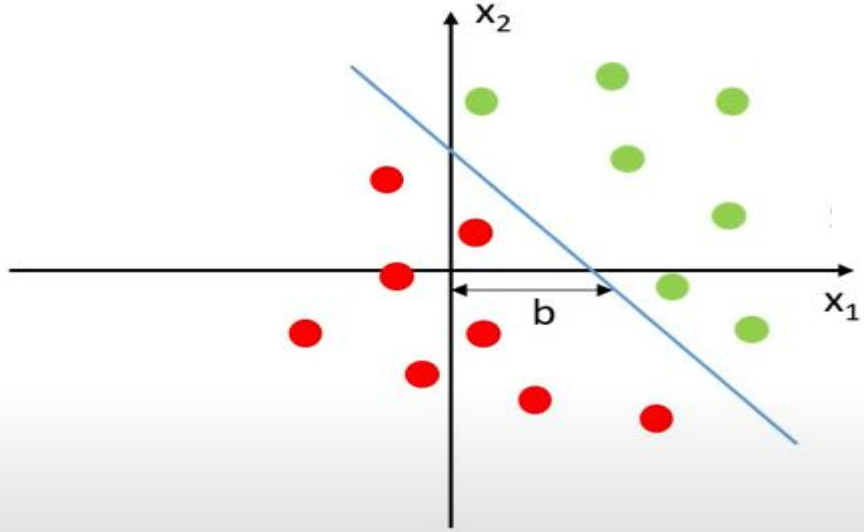
Inference: For all the points which lie in the right side of the hyperplane, $w^T x$ value will be Negative

Suppose the hyper plan not passes through the origin …?????

Solution : Add bias value (b) to the projection equation

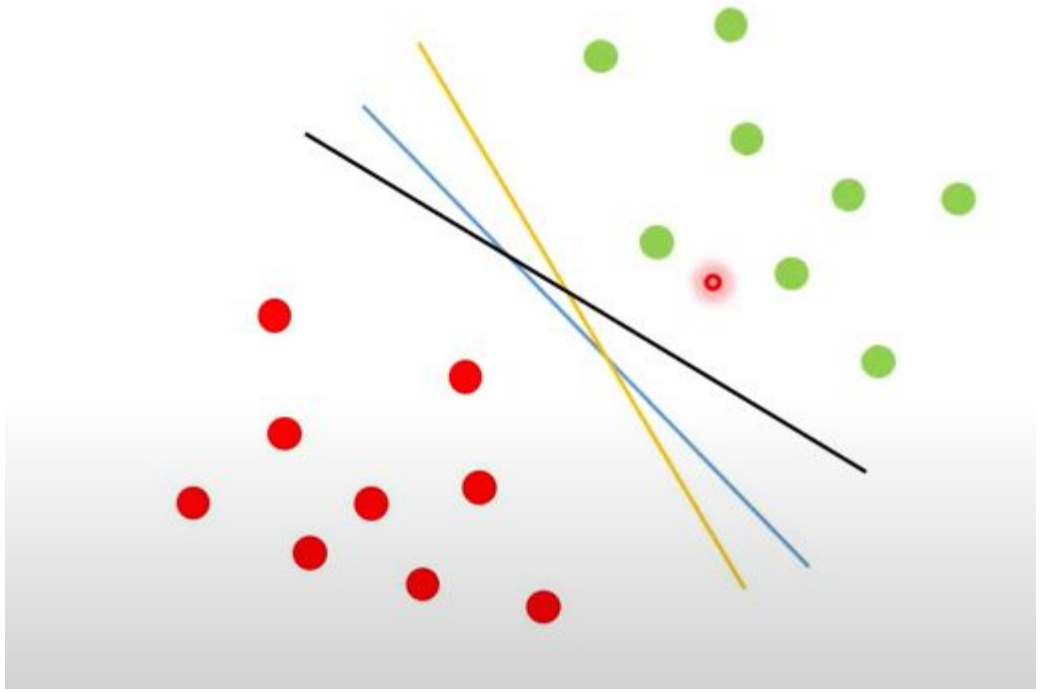


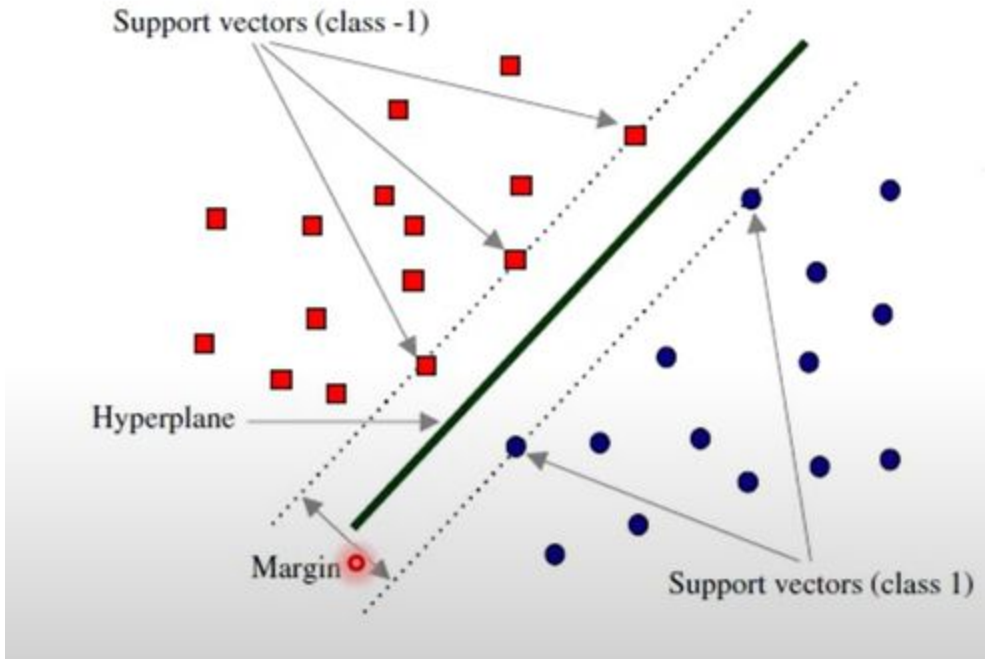
$w^T x = \text{Label}$



$w^T x + b = \text{Label}$

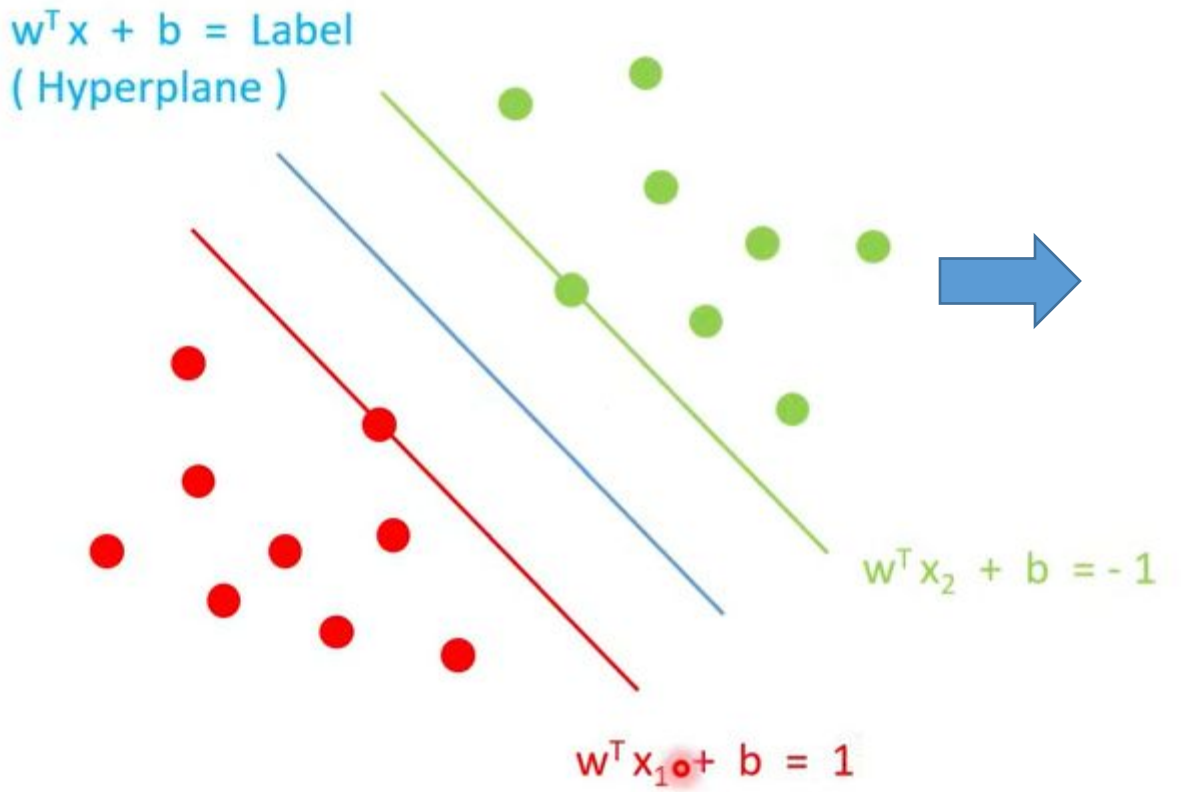
Which is the best Hyperplane?





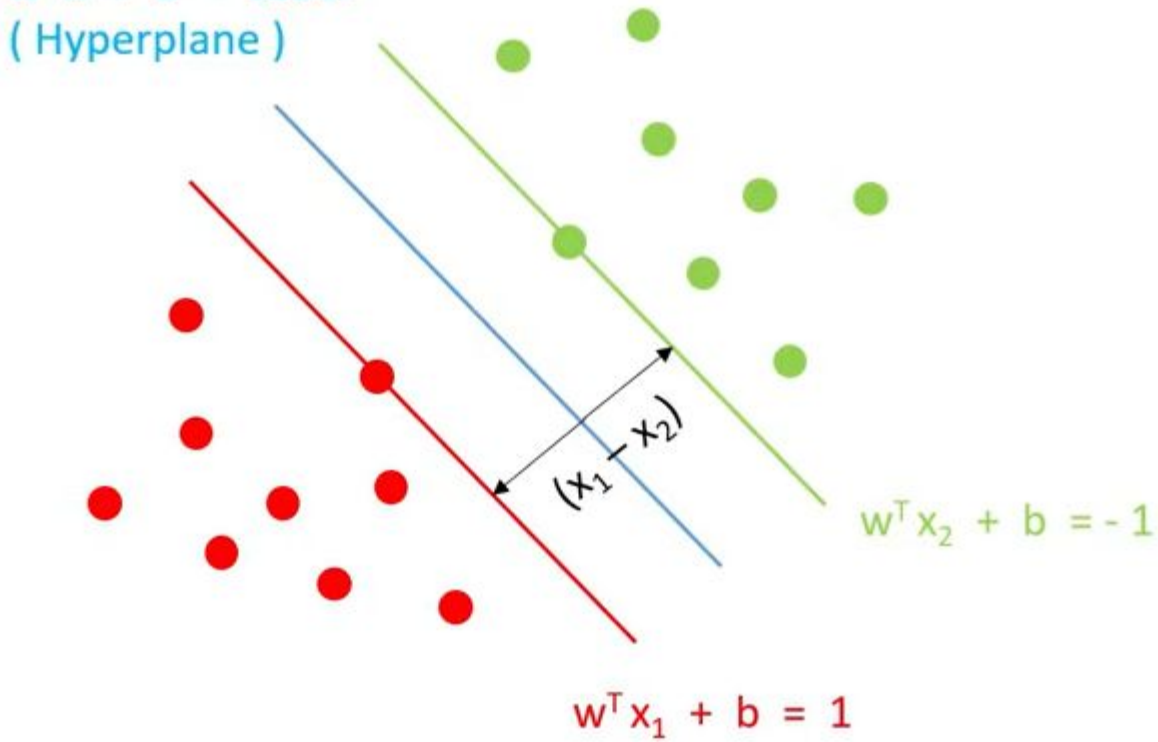
Select support vector from data points
and
Increase the distance between the margin

Optimization for Maximum margin:



To find distance between 2 margin

Optimization for Maximum margin:



$$w^T x_1 + b = 1$$

$$(-)\ w^T x_2 + b = -1$$

$$w^T (x_1 - x_2) = 2$$

Divide by $||w||$ (magnitude of the vector)

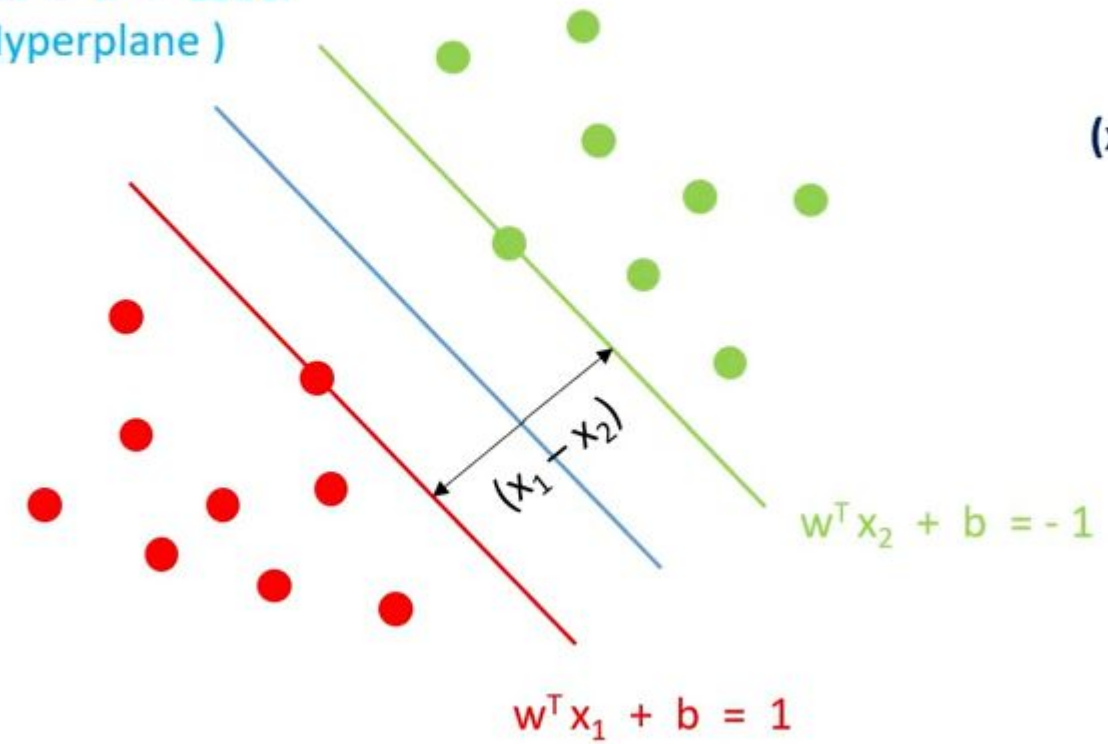
$$\frac{w^T}{||w||}(x_1 - x_2) = \frac{2}{||w||}$$

$$\mathbf{(x_1 - x_2) = \frac{2}{||w||}}$$ (margin)

Optimization for Maximum margin:

$$y_i = \begin{cases} -1, & w^T x_1 + b \leq -1 \\ 1, & w^T x_1 + b \geq 1 \end{cases} \quad \text{(Label)}$$

$w^T x + b = \text{Label}$
(Hyperplane)



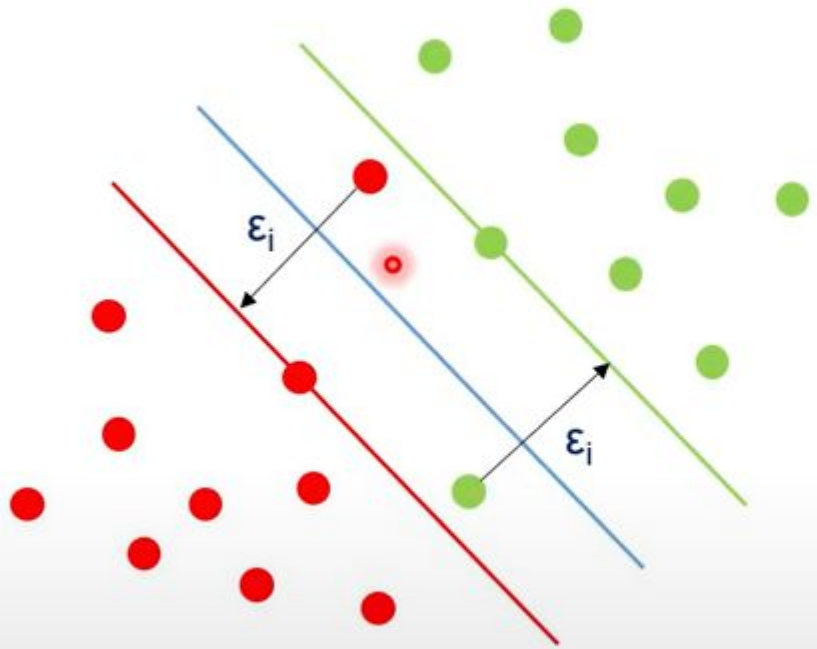
$$(x_1 - x_2) = \frac{2}{||w||} \quad \text{(margin)}$$

Sic

$$\max \left(\frac{2}{||w||} \right) \quad \text{Such that,}$$

$$y_i = \begin{cases} -1, & w^T x_1 + b \leq -1 \\ 1, & w^T x_1 + b \geq 1 \end{cases}$$

Maximum margin without overfitting:



$$\max \left(\frac{2}{||w||} \right) \quad \text{Such that,}$$

$$y_i = \begin{cases} -1, & w^T x_1 + b \leq -1 \\ 1, & w^T x_1 + b \geq 1 \end{cases}$$

$$\min \left(\frac{||w||}{2} \right) + c * \sum \varepsilon_i$$

c --> Number of errors

ε_i --> Error magnitude

Support Vector Machines

We want to maximize: $\text{Margin} = \frac{2}{\| \vec{w} \|^2}$

– Which is equivalent to minimizing: $L(w) = \frac{\| \vec{w} \|^2}{2}$

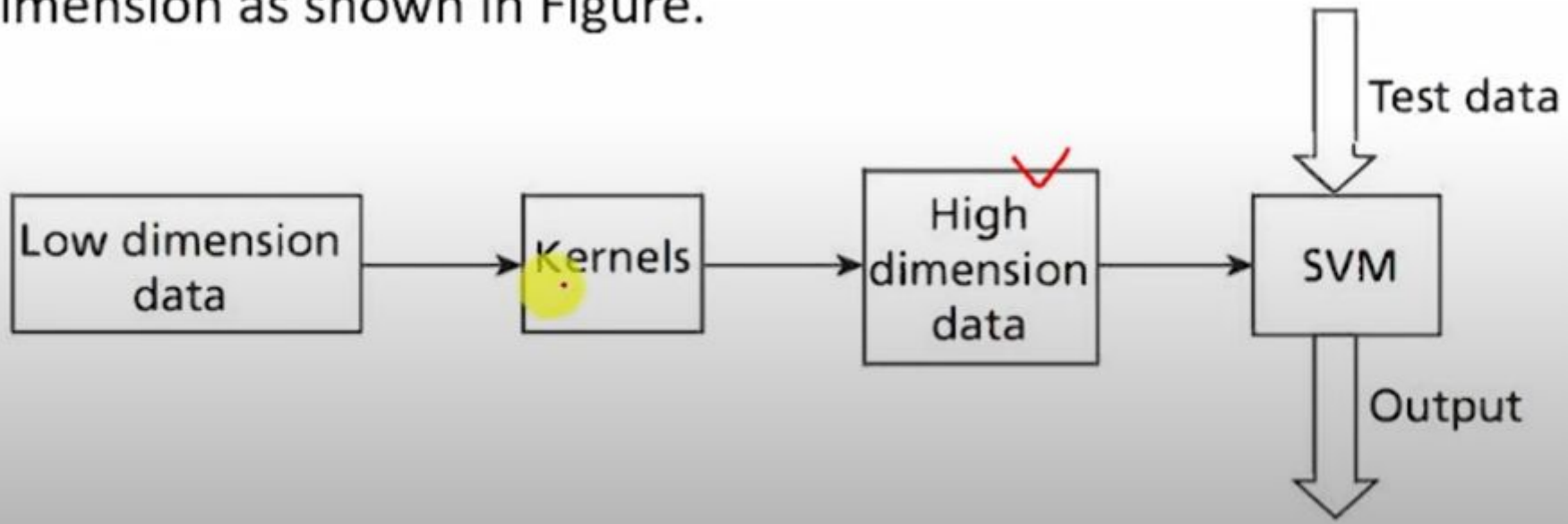
– But subjected to the following constraints:

$$f(\vec{x}_i) = \begin{cases} 1 & \text{if } \vec{w} \bullet \vec{x}_i + b \geq 1 \\ -1 & \text{if } \vec{w} \bullet \vec{x}_i + b \leq -1 \end{cases}$$

- This is a constrained optimization problem
  - Numerical approaches to solve it (e.g., quadratic programming)

Kernel Trick in Support Vector Machine

- **What is a Kernel?**
- Kernels are a set of functions used to transform data from lower dimension to higher dimension and to manipulate data using dot product at higher dimensions.
- The use of kernels is to apply transformation to data and perform classification at the higher dimension as shown in Figure.



Kernel Trick in Support Vector Machine

- For example, one mapping function $\emptyset: \boldsymbol{R^2} \rightarrow \boldsymbol{R^3}$ used to transform a 2D data to 3D data is given as follows:

$$\emptyset(x, y) = (x^2, \sqrt{2}xy, y^2)$$

- Consider a point (2, 3) in 2D space, if you apply above mapping function we can convert it into 3D space and it looks like this,
- Here $x = 2\ and\ y = 3,$
- Hence point in 3D space is:

$$(2^2, \sqrt{2} * 2 * 3, 3^2) = (4, 6\sqrt{2}, 9)$$

Kernel Trick in Support Vector Machine

- **Kernel Trick for 2nd degree Polynomial Mapping**

$$\emptyset(x, y) = (x^2, \sqrt{2}xy, y^2)$$

$$\phi(\mathbf{a})^T \cdot \phi(\mathbf{b}) = \begin{pmatrix} a_1^2 \\ \sqrt{2}\,a_1 a_2 \\ a_2^2 \end{pmatrix}^T \cdot \begin{pmatrix} b_1^2 \\ \sqrt{2}\,b_1 b_2 \\ b_2^2 \end{pmatrix} = a_1^2 b_1^2 + 2a_1 b_1 a_2 b_2 + a_2^2 b_2^2$$

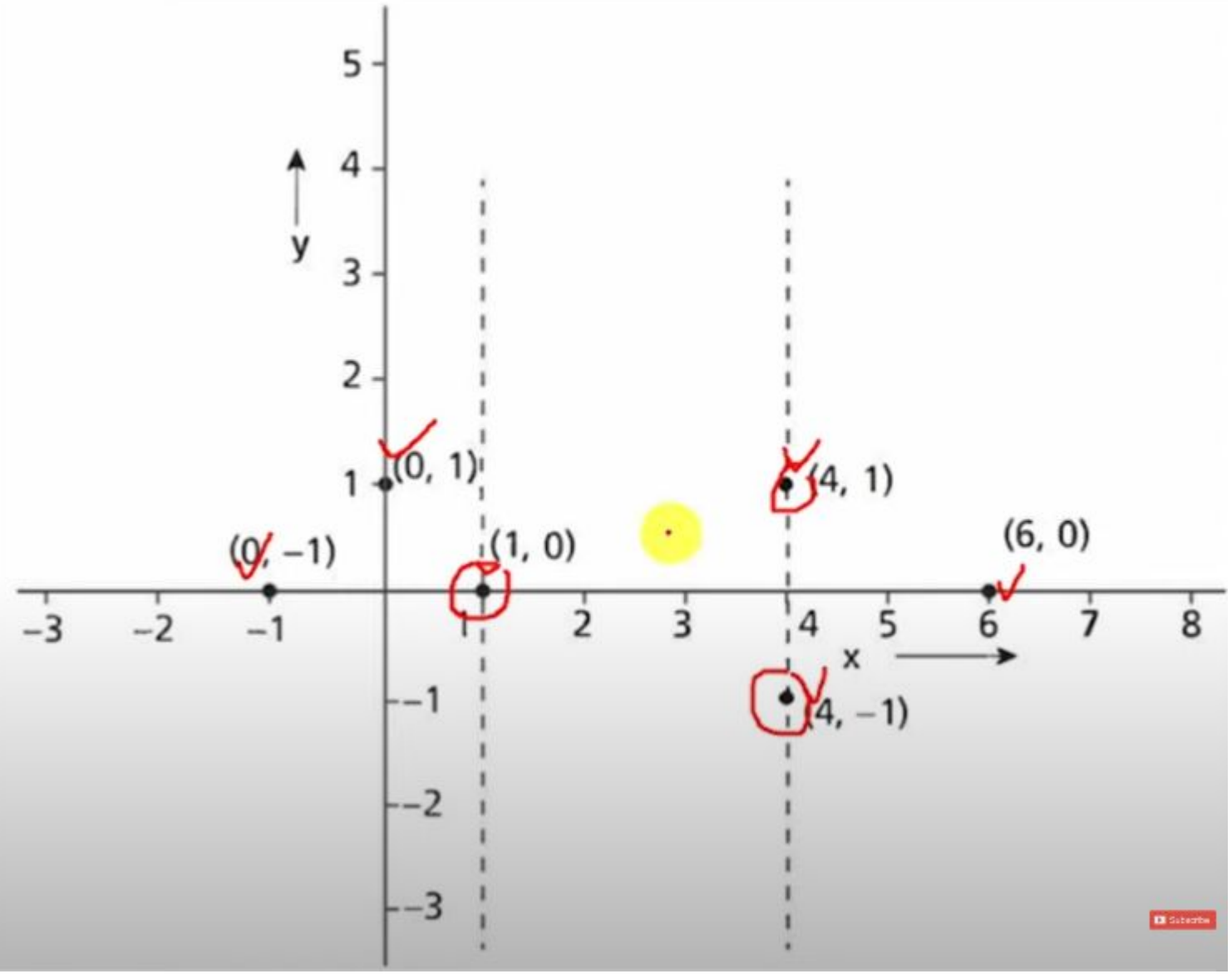
$$= (a_1 b_1 + a_2 b_2)^2 = \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}^T \cdot \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \right)^2 = \left(\mathbf{a}^T \cdot \mathbf{b}\right)^2$$

Support Vector Machine – Solved Example

- Points (4, 1), (4, -1) and (6, 0) belong to class positive and
- points (1, 0), (0, 1) and (0, -1) belong to negative class.
- Draw an optimal hyperplane to classify the points.

Support Vector Machine – Solved Example

- Points (4, 1), (4, -1) and (6, 0) belong to class positive
- points (1, 0), (0, 1) and (0, -1) belong to negative class.



$$s_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, s_2 = \begin{pmatrix} 4 \\ 1 \end{pmatrix}, s_3 = \begin{pmatrix} 4 \\ -1 \end{pmatrix}$$

- The augmented vector can be obtained by adding the bias given as follows:

$$\tilde{s}_1 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \tilde{s}_2 = \begin{pmatrix} 4 \\ 1 \\ 1 \end{pmatrix}, \tilde{s}_3 = \begin{pmatrix} 4 \\ -1 \\ 1 \end{pmatrix}$$

From these, a set of three equations can be obtained based on these three support vectors as follows:

$$\alpha_1 \tilde{s}_1 \tilde{s}_1 + \alpha_2 \tilde{s}_2 \tilde{s}_1 + \alpha_3 \tilde{s}_3 \tilde{s}_1 = -1$$
$$\alpha_1 \tilde{s}_1 \tilde{s}_2 + \alpha_2 \tilde{s}_2 \tilde{s}_2 + \alpha_3 \tilde{s}_3 \tilde{s}_2 = +1$$
$$\alpha_1 \tilde{s}_1 \tilde{s}_3 + \alpha_2 \tilde{s}_2 \tilde{s}_3 + \alpha_3 \tilde{s}_3 \tilde{s}_3 = +1$$

Support Vector Machine – Solved Example

$$\alpha_1 \begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix} + \alpha_2 \begin{pmatrix}4\\1\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix} + \alpha_3 \begin{pmatrix}4\\-1\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix}$$

$$= 2\alpha_1 + 5\alpha_2 + 5\alpha_3 = -1$$

$$\tilde{s}_1 = \begin{pmatrix}1\\0\\1\end{pmatrix}, \tilde{s}_2 = \begin{pmatrix}4\\1\\1\end{pmatrix}, \tilde{s}_3 = \begin{pmatrix}4\\-1\\1\end{pmatrix}$$

$$\alpha_1 \begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}4\\1\\1\end{pmatrix} + \alpha_2 \begin{pmatrix}4\\1\\1\end{pmatrix}\begin{pmatrix}4\\1\\1\end{pmatrix} + \alpha_3 \begin{pmatrix}4\\-1\\1\end{pmatrix}\begin{pmatrix}4\\1\\1\end{pmatrix}$$

$$= 5\alpha_1 + 18\alpha_2 + 16\alpha_3 = +1$$

$$\alpha_1 \begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}4\\-1\\1\end{pmatrix} + \alpha_2 \begin{pmatrix}4\\1\\1\end{pmatrix}\begin{pmatrix}4\\-1\\1\end{pmatrix} + \alpha_3 \begin{pmatrix}4\\-1\\1\end{pmatrix}\begin{pmatrix}4\\-1\\1\end{pmatrix}$$

$$= 5\alpha_1 + 16\alpha_2 + 18\alpha_3 = +1$$

$$\alpha_1 \tilde{s}_1 \tilde{s}_1 + \alpha_2 \tilde{s}_2 \tilde{s}_1 + \alpha_3 \tilde{s}_3 \tilde{s}_1 = -1$$

$$\alpha_1 \tilde{s}_1 \tilde{s}_2 + \alpha_2 \tilde{s}_2 \tilde{s}_2 + \alpha_3 \tilde{s}_3 \tilde{s}_2 = +1$$

$$\alpha_1 \tilde{s}_1 \tilde{s}_3 + \alpha_2 \tilde{s}_2 \tilde{s}_3 + \alpha_3 \tilde{s}_3 \tilde{s}_3 = +1$$

Support Vector Machine – Solved Example

$$\alpha_1 \begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix} + \alpha_2 \begin{pmatrix}4\\1\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix} + \alpha_3 \begin{pmatrix}4\\-1\\1\end{pmatrix}\begin{pmatrix}1\\0\\1\end{pmatrix}$$

$$= 2\alpha_1 + 5\alpha_2 + 5\alpha_3 = -1$$

$$\alpha_1 \begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}4\\1\\1\end{pmatrix} + \alpha_2 \begin{pmatrix}4\\1\\1\end{pmatrix}\begin{pmatrix}4\\1\\1\end{pmatrix} + \alpha_3 \begin{pmatrix}4\\-1\\1\end{pmatrix}\begin{pmatrix}4\\1\\1\end{pmatrix}$$

$$= 5\alpha_1 + 18\alpha_2 + 16\alpha_3 = +1$$

$$\alpha_1 \begin{pmatrix}1\\0\\1\end{pmatrix}\begin{pmatrix}4\\-1\\1\end{pmatrix} + \alpha_2 \begin{pmatrix}4\\1\\1\end{pmatrix}\begin{pmatrix}4\\-1\\1\end{pmatrix} + \alpha_3 \begin{pmatrix}4\\-1\\1\end{pmatrix}\begin{pmatrix}4\\-1\\1\end{pmatrix}$$

$$= 5\alpha_1 + 16\alpha_2 + 18\alpha_3 = +1$$

Solving these three simultaneous equations with three unknowns yields the values:

$$\alpha_1 = -3$$

$$\alpha_2 = +1$$

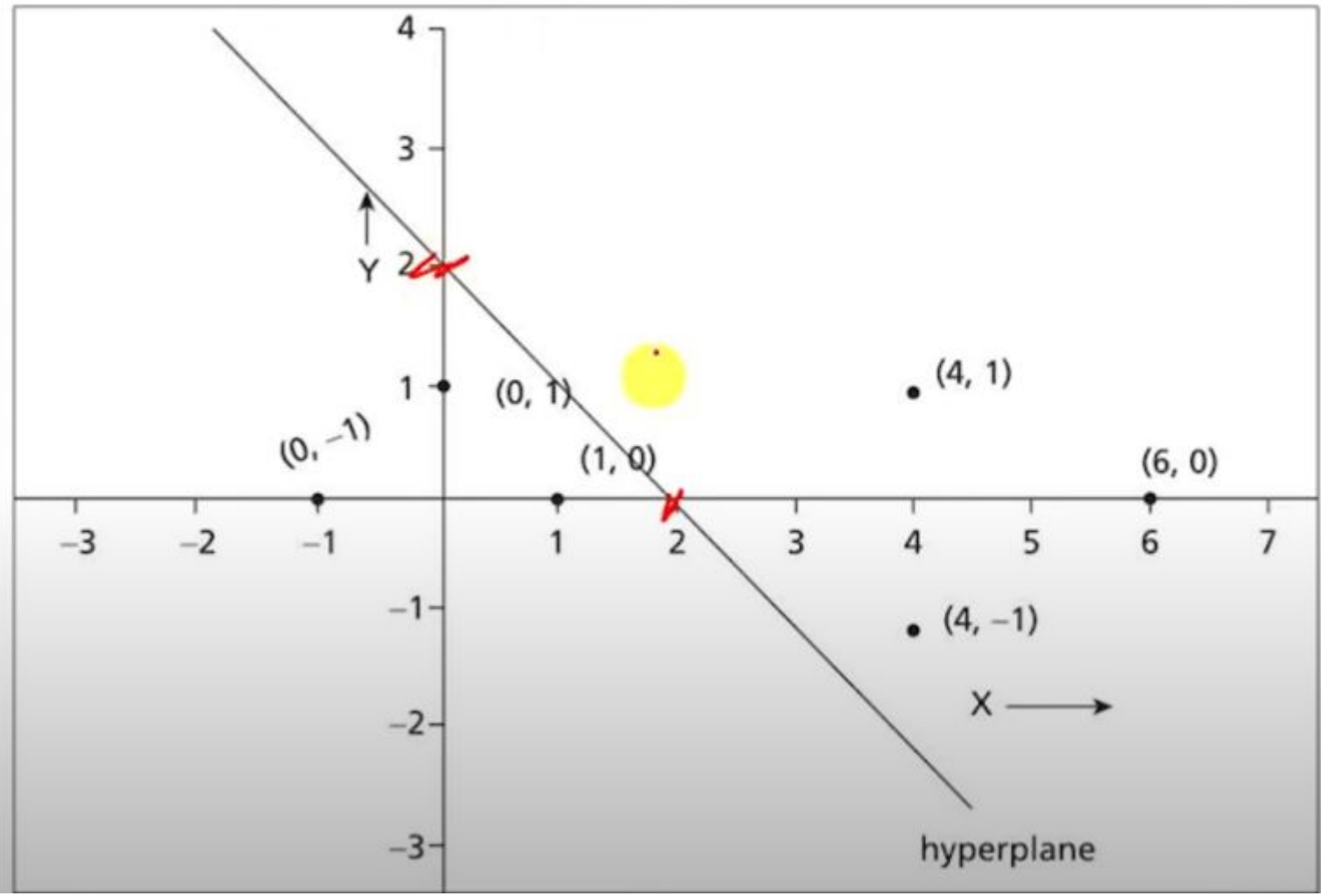
$$\alpha_3 = 0$$

The optimal Hyperplane is given as:

$$w = \sum_1^3 \alpha_i \times \tilde{s}_i$$

$$= -3 \times \begin{pmatrix}1\\0\\1\end{pmatrix} + 1 \times \begin{pmatrix}4\\1\\1\end{pmatrix} + 0 \times \begin{pmatrix}4\\-1\\1\end{pmatrix} = \begin{pmatrix}1\\1\\-2\end{pmatrix}$$

The hyperplane is (1, 1) with an offset -2.



Advantages of SVM

- The main strength of SVM is that they work well even when the number of SVM features is much larger than the number of instances.
- It can work on datasets with huge feature space, such is the case in spam filtering, where a large number of words are the potential signifiers of a message being spam.
- Even when the optimal decision boundary is a nonlinear curve, the SVM transforms the variables to create new dimensions such that the representation of the classifier is a linear function of those transformed dimensions of the data.
- SVMs are conceptually easy to understand. They create an easy-to-understand linear classifier.
- SVMs are now available with almost all data analytics toolsets.

Disadvantages of SVM

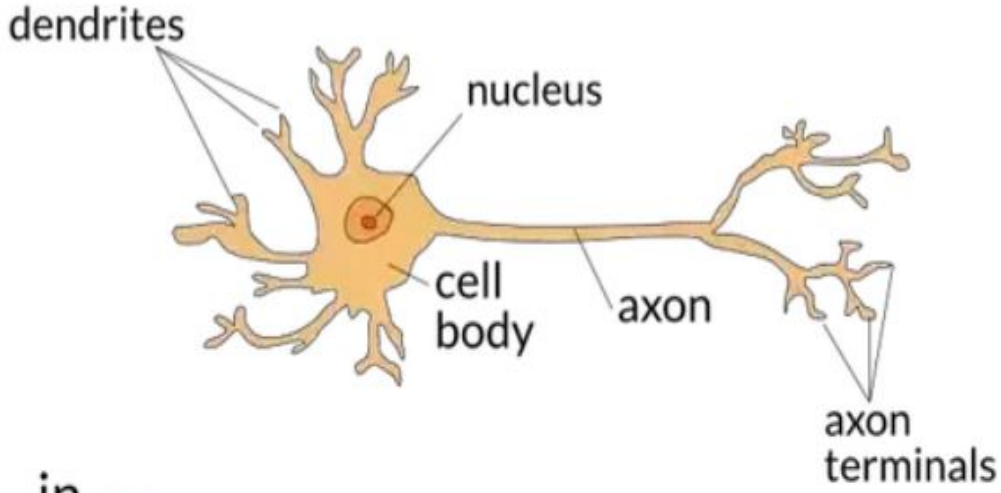
- The SVM technique has two major constraints
  - It works well only with real numbers, i.e., all the data points in all the dimensions must be defined by numeric values only,
  - It works only with binary classification problems. One can make a series of cascaded SVMs to get around this constraint.
- Training the SVMs is an inefficient and time consuming process, when the data is large.
- It does not work well when there is much noise in the data, and thus has to compute soft margins.
- The SVMs will also not provide a probability estimate of classification, i.e., the confidence level for classifying an instance.

Applications of SVMs

1. Classification
2. Regression analysis
3. Pattern recognition
4. Outliers detection.
5. Relevance based applications

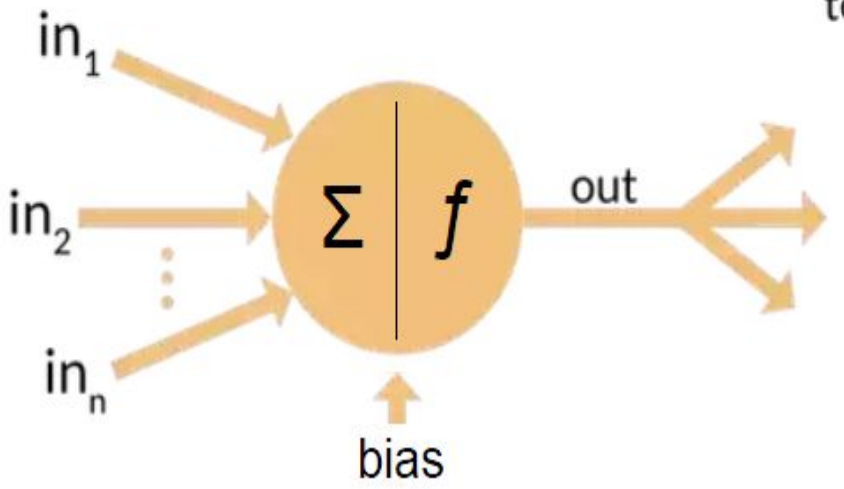
Artificial Neural Network

Artificial Neural Networks also known as Neural Networks, inspired from the neural networks of the human brain is a component of Artificial Intelligence.



Properties of artificial neural nets (ANN's):

Many neuron-like threshold switching units
Many weighted interconnections among units
Highly parallel, distributed process
Emphasis on tuning weights automatically

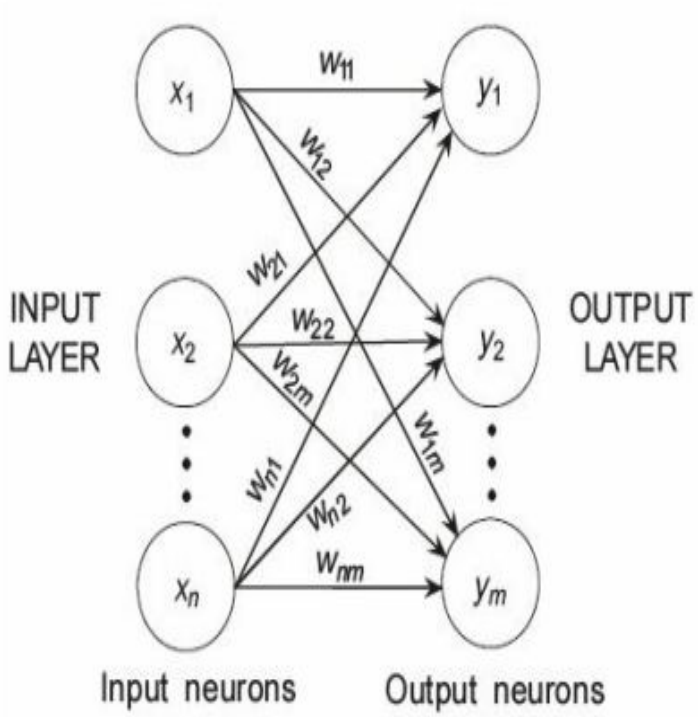


When to Consider Neural Networks

- Input is high-dimensional discrete or real-valued (e.g. raw sensor input)
- Output is discrete or real valued
- Output is a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human readability of result is unimportant

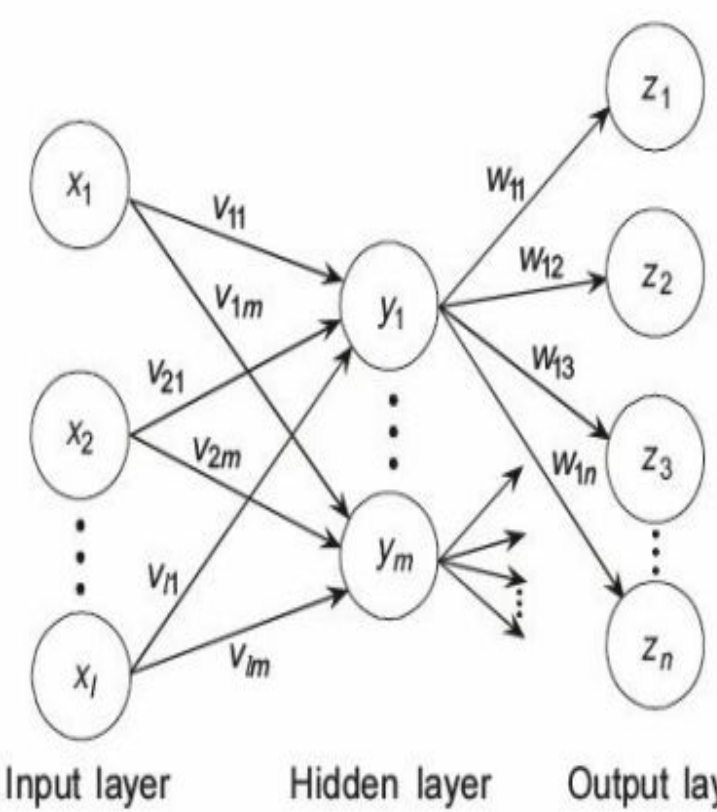
NEURAL NETWORK ARCHITECTURES

Single Layer Feed forward Network



x_i: Input neurons
y_j: Output neurons
w_{ij}: Weights

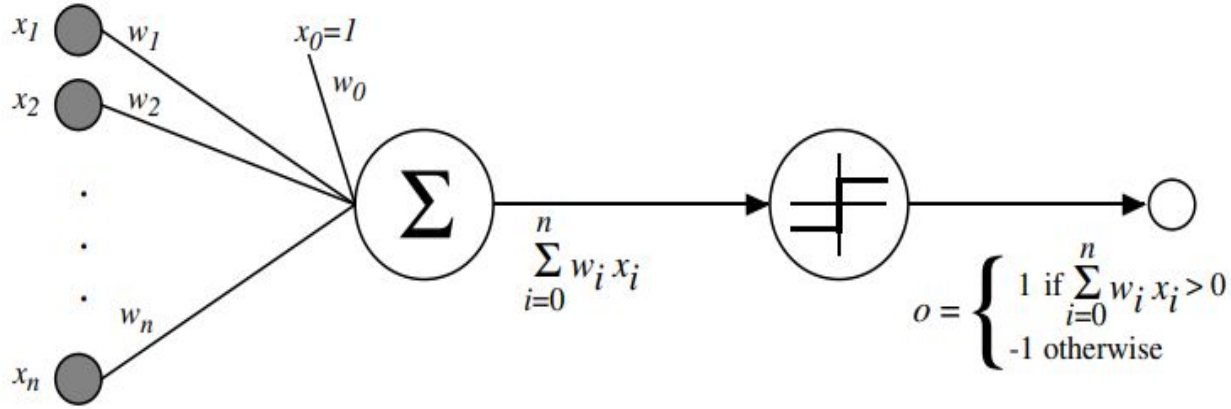
Multilayer Feed forward Network



x_i : Input neurons
y_j : Hidden neurons
z_k : Output neurons
v_{ij} : Input hidden layer weights
w_{jk} : Output hidden layer weights

Perceptron

ANN system is based on a unit called a perceptron

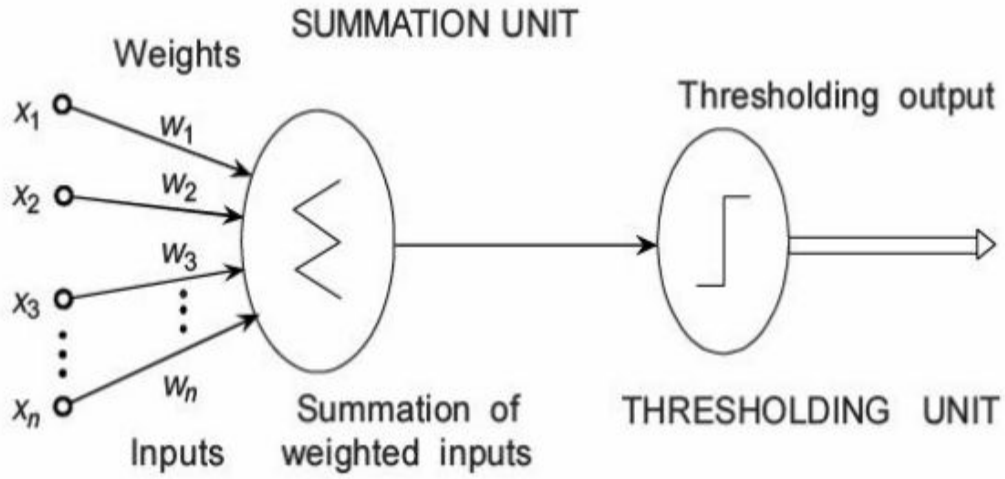


$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Sometimes we'll use simpler vector notation:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

$\boldsymbol{w_i}$ is a real-valued constant, or weight, that determines the contribution of input $\boldsymbol{x_i}$ to the perceptron output



Perceptron training rule

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

Where:

- $t = c(\vec{x})$ is target value
- o is perceptron output
- η is small constant (e.g., .1) called *learning rate*

Suppose the training example is correctly classified already by the perceptron

(t-o) is zero, making Δw_i

For example, if ***xi > 0,*** then increasing w_i will bring the perceptron closer to correctly classifying the data

Can prove it will converge

- If training data is linearly separable
- and η sufficiently small

Gradient Descent and the Delta Rule

perceptron rule finds a successful weight vector when the training examples are linearly separable, it can fail to converge if the examples are not linearly separable.

Delta rule: is designed to not linearly separable.

To understand, consider simpler *linear unit*, where

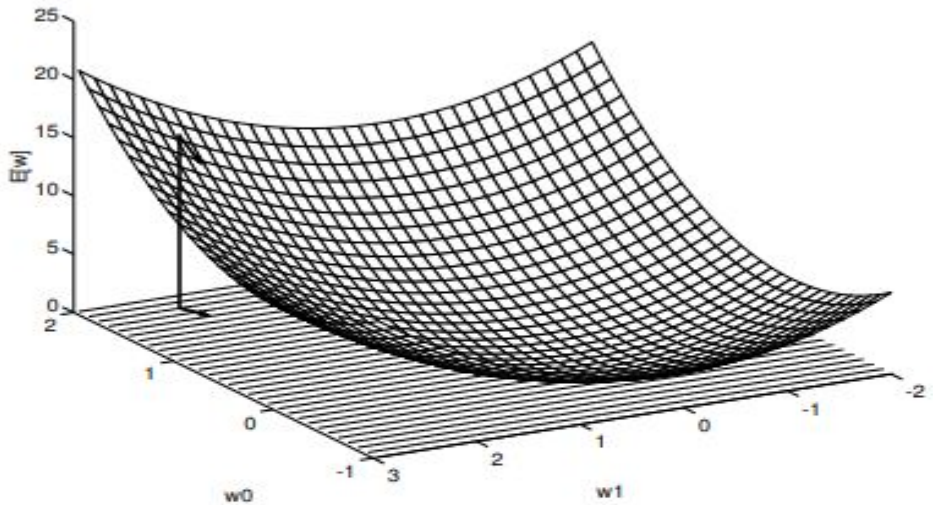
$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

Let's learn w_i's that minimize the squared error

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Where D is set of training examples

Gradient Descent



Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\
&= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d) \\
\frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d)(-x_{i,d})
\end{aligned}$$

Gradient Descent

GRADIENT-DESCENT($training\_examples, \eta$)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, *where* $\vec{x}$ *is the vector of input values, and* t *is the target output value.* η *is the learning rate (e.g., .05).*

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in $training\_examples$, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$

  - For each linear unit weight w_i, Do

$$w_i \leftarrow w_i + \Delta w_i$$

Perceptron training rule guaranteed to succeed if

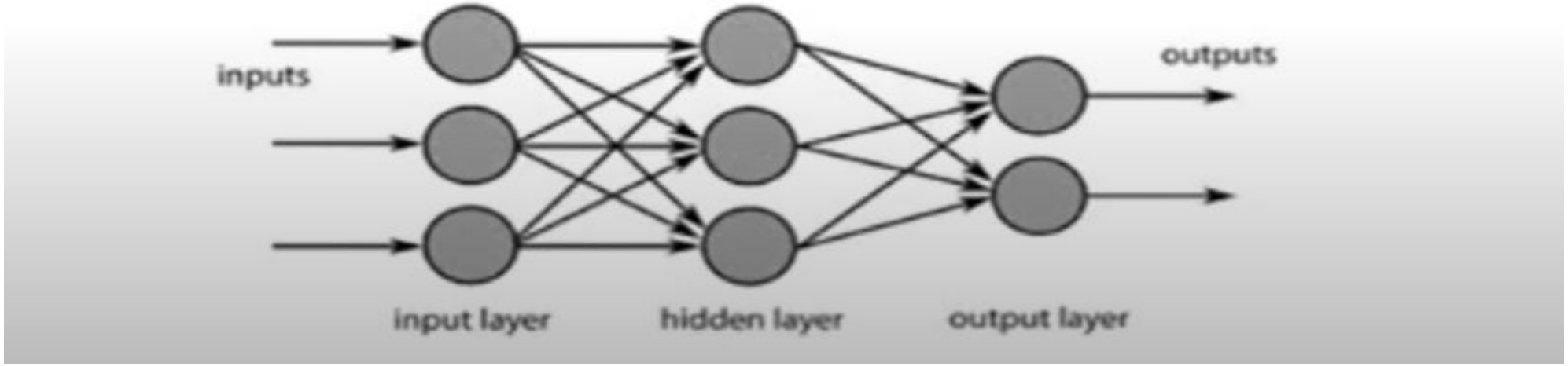
- Training examples are linearly separable
- Sufficiently small learning rate η

Linear unit training rule uses gradient descent

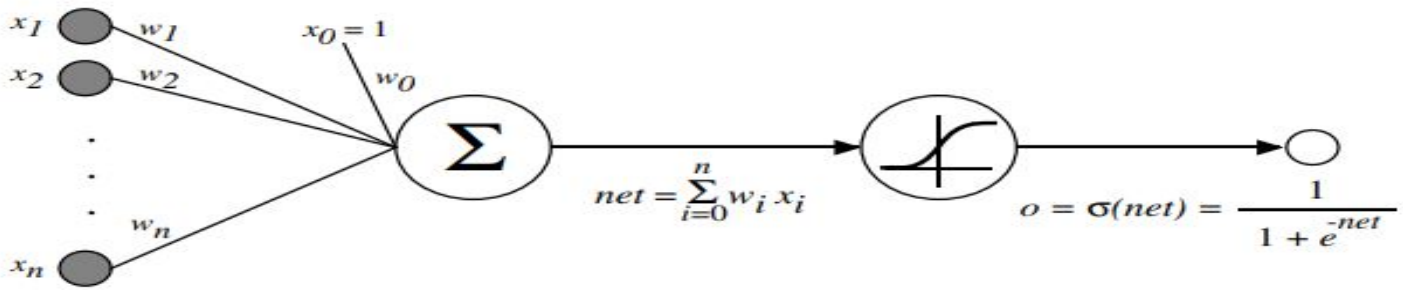
- Guaranteed to converge to hypothesis with minimum squared error
- Given sufficiently small learning rate η
- Even when training data contains noise
- Even when training data not separable by H

Multi-Layer Networks

- A Multi layer perceptron can classify non linear separable problems.
- A Multilayer network has one or more hidden layers.



Sigmoid Unit



$\sigma(x)$ is the sigmoid function

$$\frac{1}{1 + e^{-x}}$$

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

We can derive gradient decent rules to train

- One sigmoid unit
- *Multilayer networks* of sigmoid units $\rightarrow$ Backpropagation

MLP: Some Preliminaries

- **The multilayer perceptron (MLP) is proposed to overcome the limitations of the perceptron**
  - That is, building a network that can ***solve nonlinear*** problems.

- **The basic features of the multilayer perceptrons:**
  - Each neuron in the network includes a nonlinear activation function that is *differentiable*.
  - The network contains one or more layers that are *hidden* from both the input and output nodes.
  - The network exhibits a high degree of *connectivity*.

Table 2.3 XOR truth table

Inputs	Inputs	Output	
0	0	0	Even parity
1	1	0	
0	1	1	Odd parity
1	0	1	

- **Architecture of a multilayer perceptron**

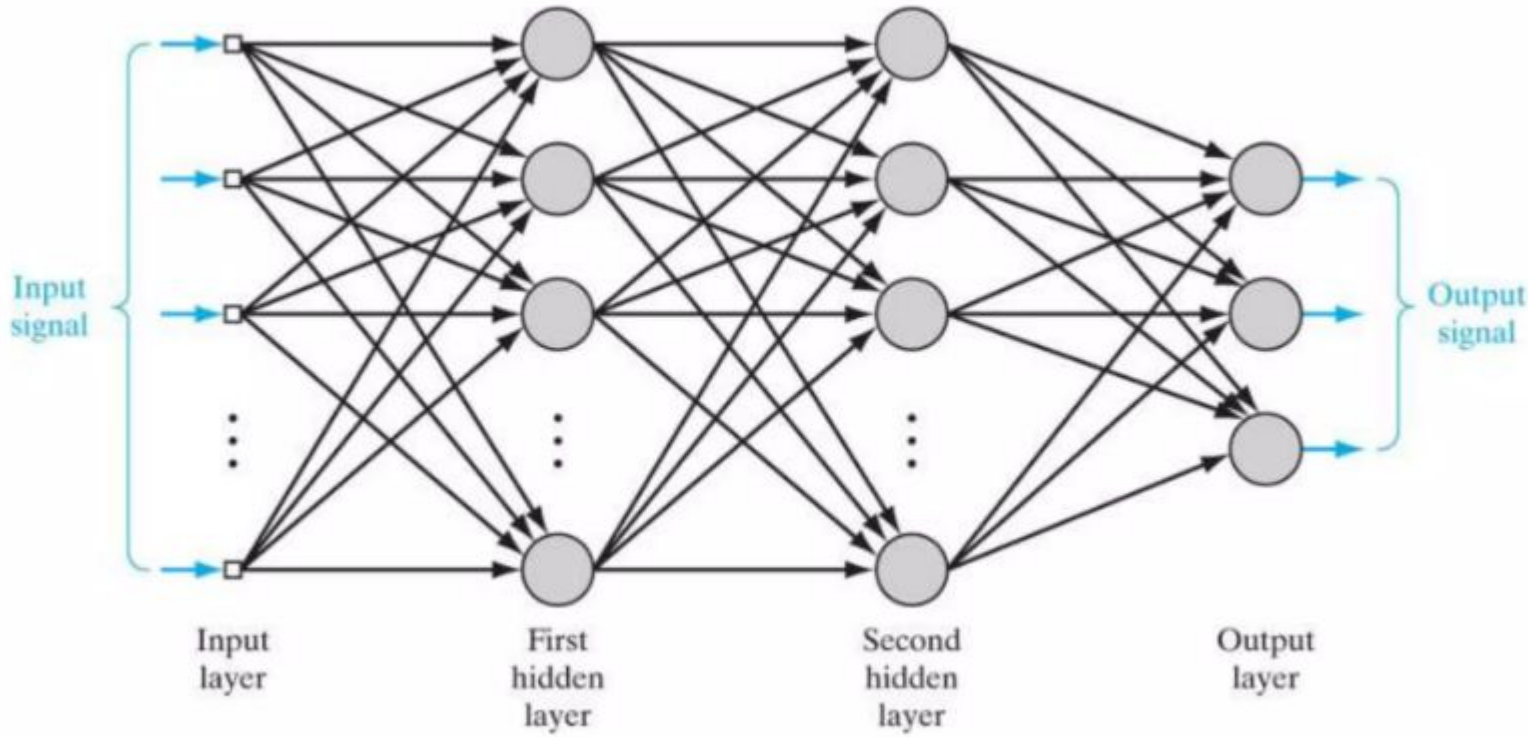
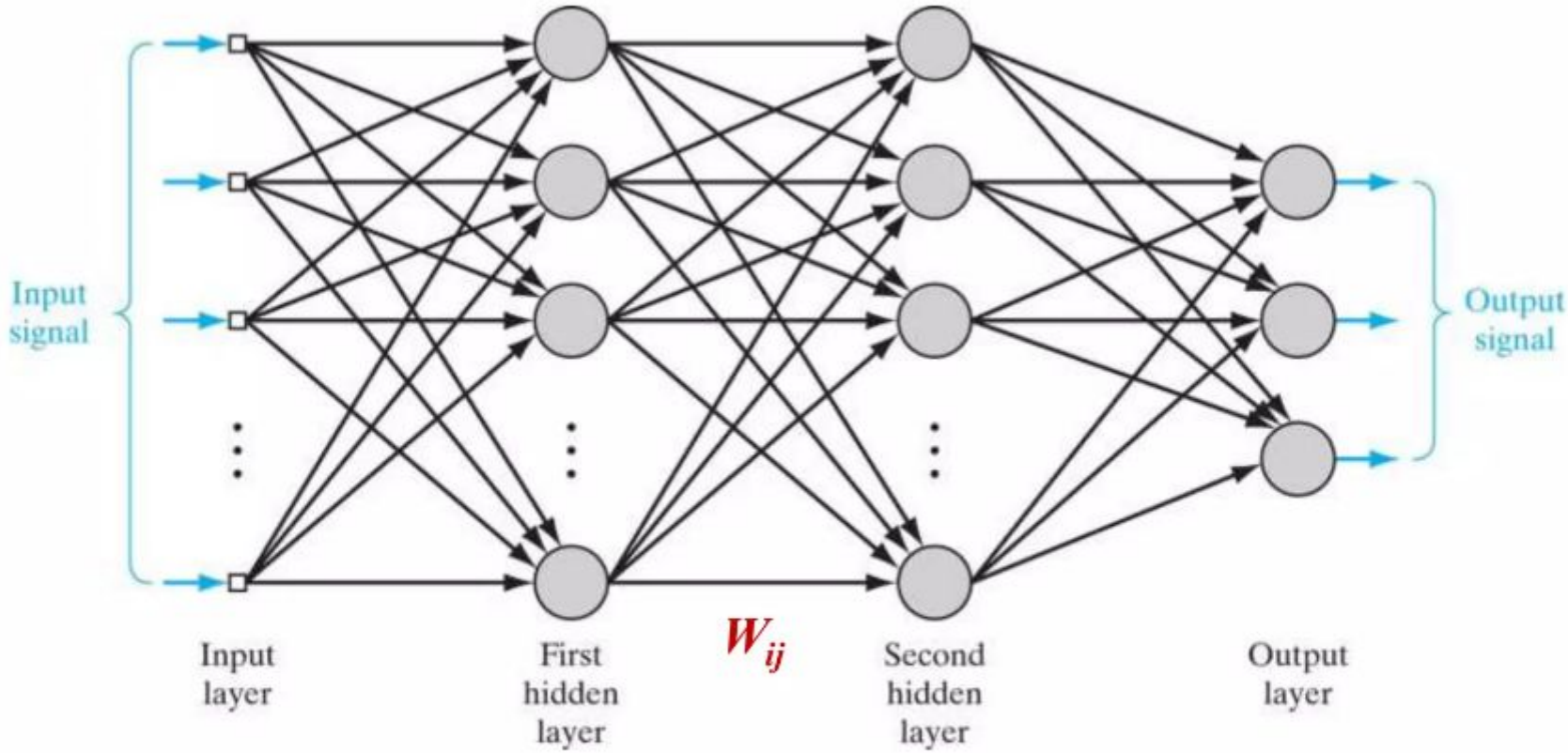


Figure 4.1 Architectural graph of a multilayer perceptron with two hidden layers.

- **Weight Dimensions**



If network has n units in layer i , m units in layer $i+1$, then the weight matrix W_{ij} will be of dimension $m \times (n+1)$.

- ***Training* of the *multilayer perceptron* proceeds in *two phases*:**
  - In the ***forward*** *phase*, the weights of the network are *fixed* and the input signal is *propagated* through the network, layer by layer, until it reaches the *output*.
  - In the ***backward*** *phase*, the error signal, which is produced by comparing the output of the network and the desired response, is *propagated* through the network, again layer by layer, but in the *backward direction*.

MLP: Some Preliminaries

- **Function Signal:**
  - is the input signal that comes in at the input end of the network, ***propagates forward*** (neuron by neuron) through the network, and emerges at the output of the network as an ***output signal***.
- **Error Signal:**
  - originate at the output neuron of the network and ***propagates backward*** (layer by layer) through the network.
- **Each *hidden* or *output neuron* computes these two signals.**

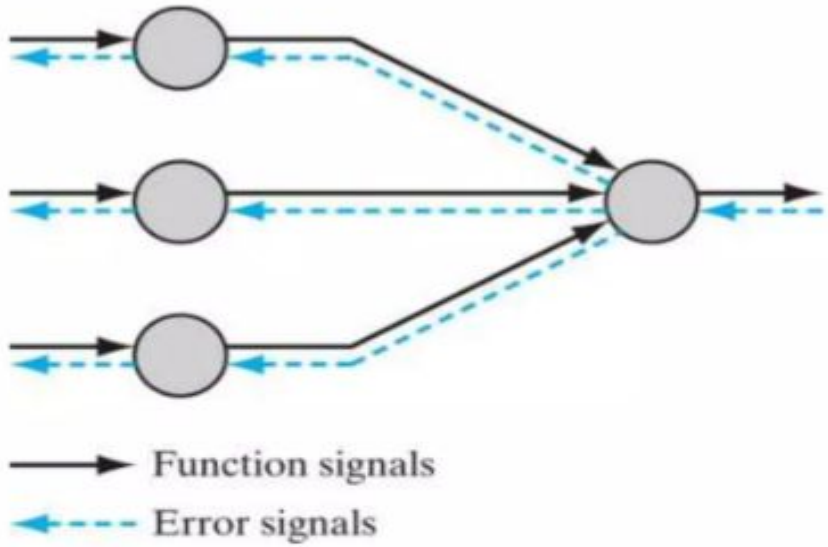


Figure 4.2 Illustration of the directions of two basic signal flows in a multilayer perceptron: forward propagation of function signals and back propagation of error signals.

- **Function of the Hidden neurons**
  - The ***hidden neurons*** play a critical role in the operation of a multilayer perceptron; they act as ***feature detectors***.
  - The ***nonlinearity*** transform the input data into a ***feature space*** in which data ***may be separated easily***.

Derivation Of The Backpropagation Rule

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

$$E_d(\vec{w}) \equiv \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

- x_{ji} = the ith input to unit j
- w_{ji} = the weight associated with the ith input to unit j
- $net_j = \sum_i w_{ji} x_{ji}$ (the weighted sum of inputs for unit j)
- o_j = the output computed by unit j
- t_j = the target output for unit j
- σ = the sigmoid function
- $outputs$ = the set of units in the final layer of the network
- $Downstream(j)$ = the set of units whose immediate inputs include the output of unit j

Case 1: Training Rule for Output Unit Weights

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = \eta \ (t_j - o_j) \ o_j(1 - o_j) x_{ji}$$

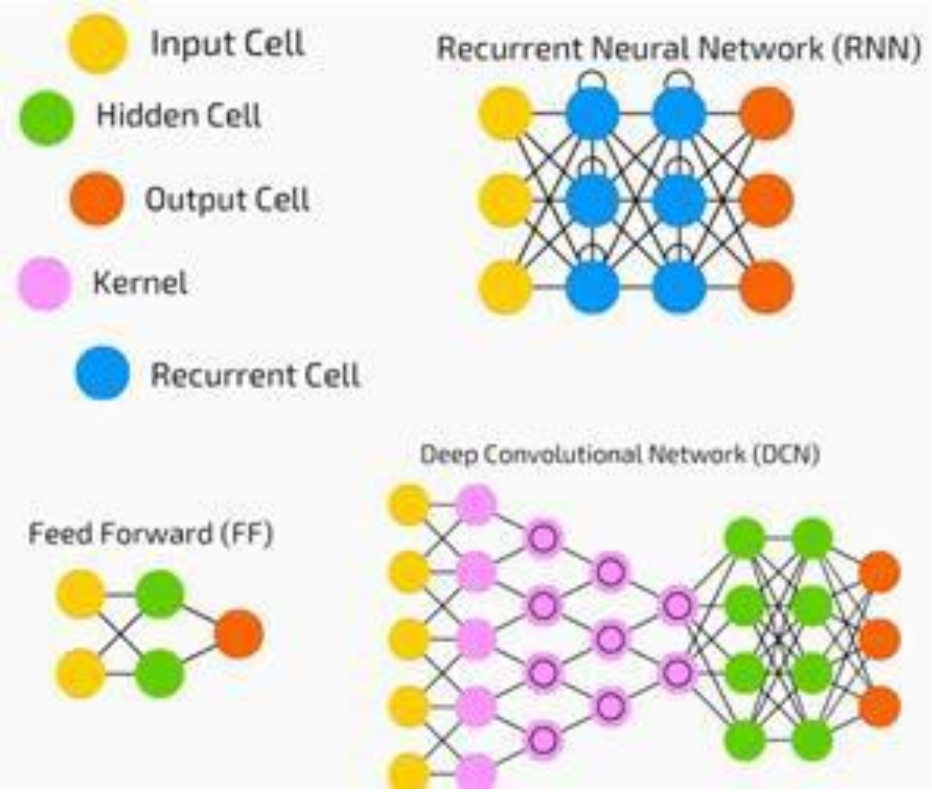
Case 2: Training Rule for Hidden Unit Weights

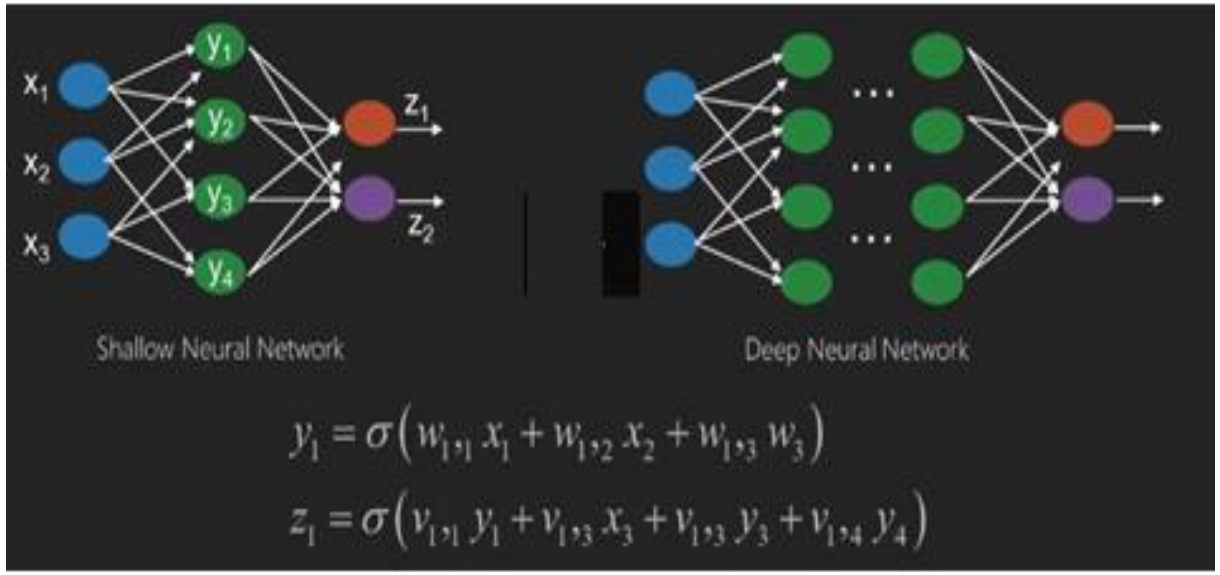
$$\delta_j = o_j(1 - o_j) \sum_{k \in Downstream(j)} \delta_k \ w_{kj}$$

$$\Delta w_{ji} = \eta \ \delta_j \ x_{ji}$$

Deep Neural Network

A deep neural network (DNN) is an ANN with multiple hidden layers between the input and output layers. Similar to shallow ANNs, DNNs can model complex non-linear relationships.





Choosing a Deep Net

- For text processing, sentiment analysis, parsing and name entity recognition, we use a recurrent net or recursive neural tensor network or RNTN;
- For any language model that operates at character level, we use the recurrent net.
- For image recognition, we use deep belief network DBN or convolutional network.
- For object recognition, we use a RNTN or a convolutional network.
- For speech recognition, we use recurrent net.

The Back-propagation Algorithm

- **We summarize the relations for the back-propagation algorithm:**
  - **First**: the correction $\Delta w_{ji}(n)$ applied to the weight connecting neuron i to neuron j is defined by the delta rule:

$$\begin{pmatrix}\text{weight}\\ \text{correction}\\ \Delta w_{ji}(n)\end{pmatrix}=\begin{pmatrix}\text{learning}-\text{rate}\\ \text{parameter}\\ \eta\end{pmatrix}\times\begin{pmatrix}\text{local}\\ \text{gradient}\\ \delta_j(n)\end{pmatrix}\times\begin{pmatrix}\text{input signal}\\ \text{of neuron j}\\ y_i(n)\end{pmatrix}$$

  - **Second**: local gradient $\delta_j(n)$ depends on neuron j :
    - Neuron j is an output node:

$$\delta_j(n)=e_j(n)\varphi'_j(v_j(n));\qquad e_j(n)=d_j(n)-y_j(n)$$

    - Neuron j is an hidden node (neuron k is output or hidden):

$$\delta_j(n)=\varphi'_j(v_j(n))\sum_k\delta_k(n)w_{kj}(n)$$

The Back-propagation Algorithm

which yields:

$$\frac{\partial \mathscr{E}_j(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi'_j(v_j(n))y_i(n)$$

Then the weight correction is given by:

$$\boxed{\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)}$$

where the *local gradient* δ_j (n) is defined by:

$$\delta_j(n) = -\frac{\partial \mathscr{E}_j(n)}{\partial v_j(n)}$$

$$= -\frac{\partial \mathscr{E}_j(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)}$$

$$\boxed{\delta_j(n) = e_j(n)\varphi'_j(v_j(n))}$$

The Activation Function

- ***Differentiability*** **is the only** ***requirement*** **that an activation function has to satisfy in the BP Algoruthm.**
  - This is required to compute the δ for each neuron.
- **Sigmoidal functions are commonly used, since they satisfy such a condition:**

  - Logistic Function

$$\varphi(v) = \frac{1}{1+\exp(-av)}, \qquad a > 0 \quad \Rightarrow \quad \varphi'(v) = \frac{a\exp(-av)}{1+\exp(-av)} = a\varphi(v)[1-\varphi(v)]$$

  - Hyperbolic Tangent Function

$$\varphi(v) = a\tanh(bv), \qquad a, b > 0 \quad \Rightarrow \quad \varphi'(v) = \frac{b}{a}[a-\varphi(v)][a+\varphi(v)]$$

The Rate of Learning

- A simple method of *increasing* the rate of learning and *avoiding instability* (for large learning rate η) is to modify the delta rule by including a *momentum* term as:

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n)$$

- where α is usually a positive number called the *momentum constant*.
- To ensure *convergence*, the momentum constant must be restricted to

$$0 \le |\alpha| < 1$$

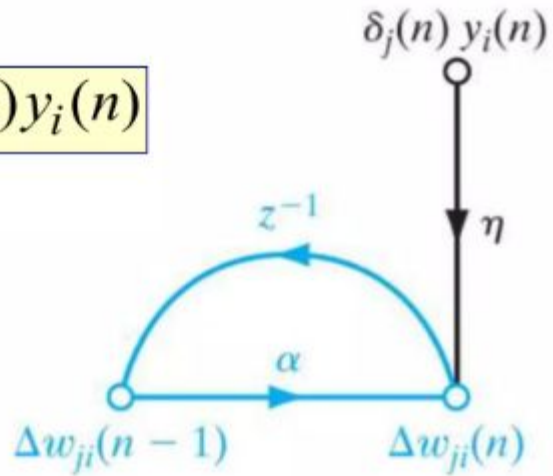


Figure 4.6 Signal-flow graph illustrating the effect of momentum constant α, which lies inside the feedback loop.

https://www.youtube.com/watch?v=tUiZ7hltWSg

https://www.slideserve.com/howard/support-vector-machines-powerpoint-ppt-presentation

https://www.youtube.com/@MaheshHuddar/search?query=SVM

https://www.youtube.com/watch?v=rhz7AGlO5fw&list=PL4gu8xQu0_5JdEqxU_FcY2x2aPte_BzF5&index=2

https://www.slideshare.net/slideshow/lecture9support-vector-machines-algorithmsml1ppt/265074445

https://www.youtube.com/watch?v=4-9HlWdZpJo

https://www.youtube.com/watch?v=VJ7WF_Dr3Os

https://www.youtube.com/watch?v=y0_Qq6fXzCs

https://www.youtube.com/watch?v=ivPoCcYfFAw

https://www.youtube.com/watch?v=ivPoCcYfFAw&list=PL4gu8xQu0_5JdEqxU_FcY2x2aPte_BzF5&index=5

https://www.youtube.com/watch?v=VJ7WF_Dr3Os&list=PL4gu8xQu0_5JdEqxU_FcY2x2aPte_BzF5&index=7