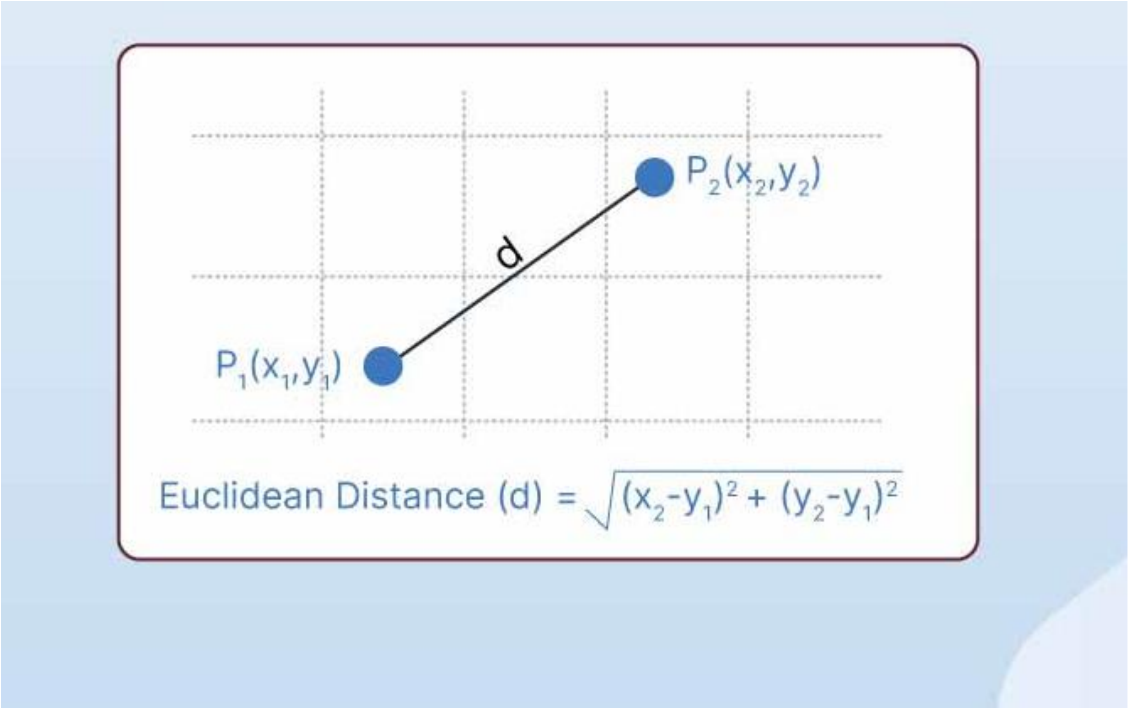
COMP30027 MACHINE LEARNING TUTORIAL

Workshop - 5

Distance Measures, KNN & SVM

- How to implement distance measures (Euclidean, Manhattan, etc.)
- How to implement K-NN
- How to implement SVM
- Design decisions involved in K-NN and SVM classification



Euclidean Distance

Formula

$\sqrt{(\Sigma(xi\ -\ y_i)^2)}$ calculates the straight-line distance between points.

Best Use

Ideal for continuous numerical features

Limitation

Highly sensitive to feature scaling. Normalization is essential.

Euclidean Distance

#	Emp	Age	Salary
1	Emp1	44	73000
2	Emp2	27	47000
3	Emp3	30	53000
4	Emp4	38	62000
5	Emp5	40	57000
6	Emp6	35	53000
7	Emp7	48	78000

Distance between Emp2 and Emp1 = $\sqrt{(27-44)^2+(47000-73000)^2}$ = 31.06

Distance between Emp2 and Emp3 = $\sqrt{(30-27)^2+(53000-47000)^2}$ = 6.70

Formula

$\sqrt{(\Sigma(xi\ -\ y_i)^2)}$ calculates the straight-line distance between points.

Best Use

Ideal for continuous numerical features

Limitation

Highly sensitive to feature scaling. Normalization is essential.

Feature Scaling Techniques

Min-Max Normalization

$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

#	Emp	Age	Salary
1	Emp1	44	73000
2	Emp2	27	47000
3	Emp3	30	53000
4	Emp4	38	62000
5	Emp5	40	57000
6	Emp6	35	53000
7	Emp7	48	78000

Normalization

Age	Normalized Age	Salary	Normalized Salary
44	0.80952381	73000	0.838709677
27	0	47000	0
30	0.142857143	53000	0.193548387
38	0.523809524	62000	0.483870968
40	0.619047619	57000	0.322580645
35	0.380952381	53000	0.193548387
48	1	78000	1
	Range 0-1		**Range 0-1**

How to calculate Normalized value?
X = 35, min = 27, max = 48 for column Age.
Xnorm(for 35) = $\frac{35-27}{48-27} = 0.3809$

After Normalization

Distance between Emp2 and Emp1 = $\sqrt{(0-.80)^2+(0-.83)^2} = 1.15$
Distance between Emp2 and Emp3 = $\sqrt{(.14-0)^2+(.19-0)^2} = 0.23$

Comparison will be more significant

Standardization

$$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation }(x)}$$

#	Emp	Age	Salary
1	Emp1	44	73000
2	Emp2	27	47000
3	Emp3	30	53000
4	Emp4	38	62000
5	Emp5	40	57000
6	Emp6	35	53000
7	Emp7	48	78000
		Mean = 37.42857 Std. Dev. = 6.883876	Mean = 60428.5714 Std. Dev = 10499.7570

Standardization

How to calculate Standardized value?
X = 35, mean = 37.42, Std. Dev. = 6.88
for column Age.
Xstd(for 35) = $\frac{35-37.42}{6.88}$ =-0.3527

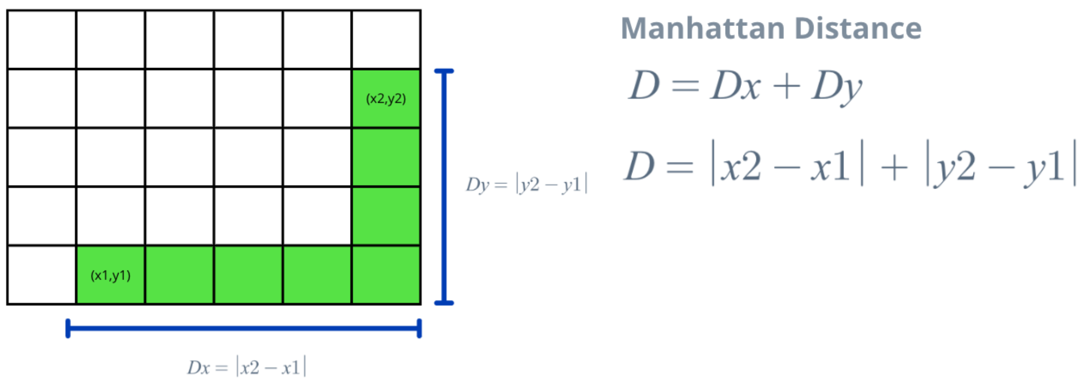
Age	Standardized Age	Salary	Standardized Salary
44	0.954611636	73000	1.197306616
27	-1.514927162	47000	-1.278941158
30	-1.079126198	53000	-0.707499364
38	0.083009708	62000	0.149663327
40	0.373543684	57000	-0.326538168
35	-0.352791257	53000	-0.707499364
48	1.535679589	78000	1.673508111
	Mean = 0 **Std. dev. = 1**		**Mean = 0** **Std. dev. = 1**

After Standardization

Distance between Emp2 and Emp1 = $\sqrt{(-1.51 - 0.95)^2+(-1.27-1.19)^2} = 3.47$
Distance between Emp2 and Emp3 = $\sqrt{(-1.07+1.51)^2+(-0.70+1.27)^2} = 0.71$

Comparison will be more significant

https://ashutoshtripathi.com/2021/06/12/what-is-feature-scaling-in-machine-learning-normalization-vs-standardization/



Manhattan Distance

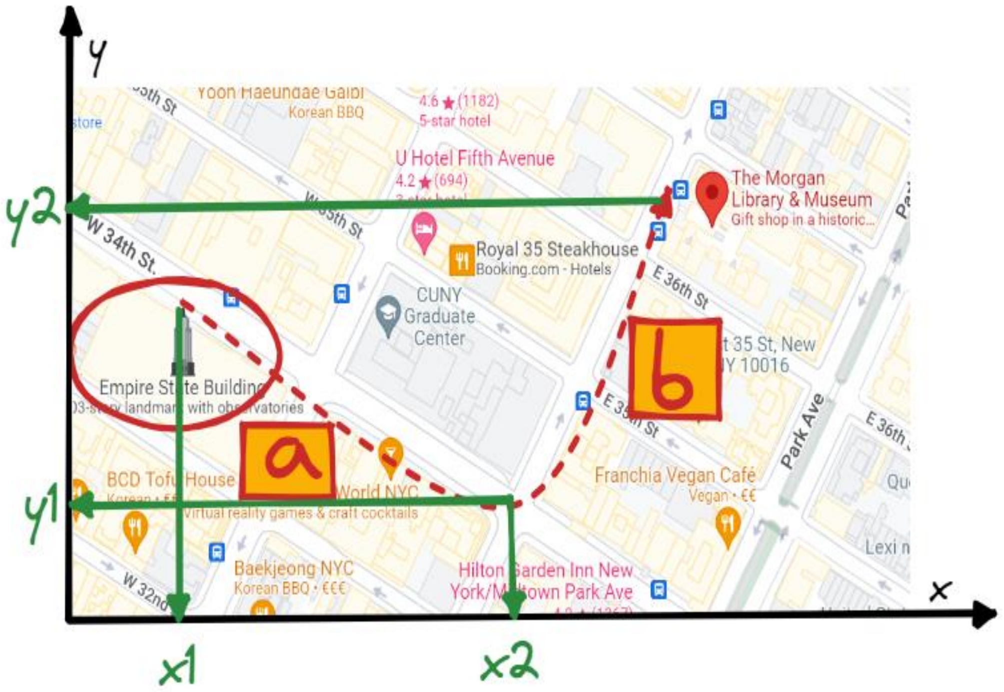
What Is It?

Manhattan distance measures the sum of absolute differences between coordinates.

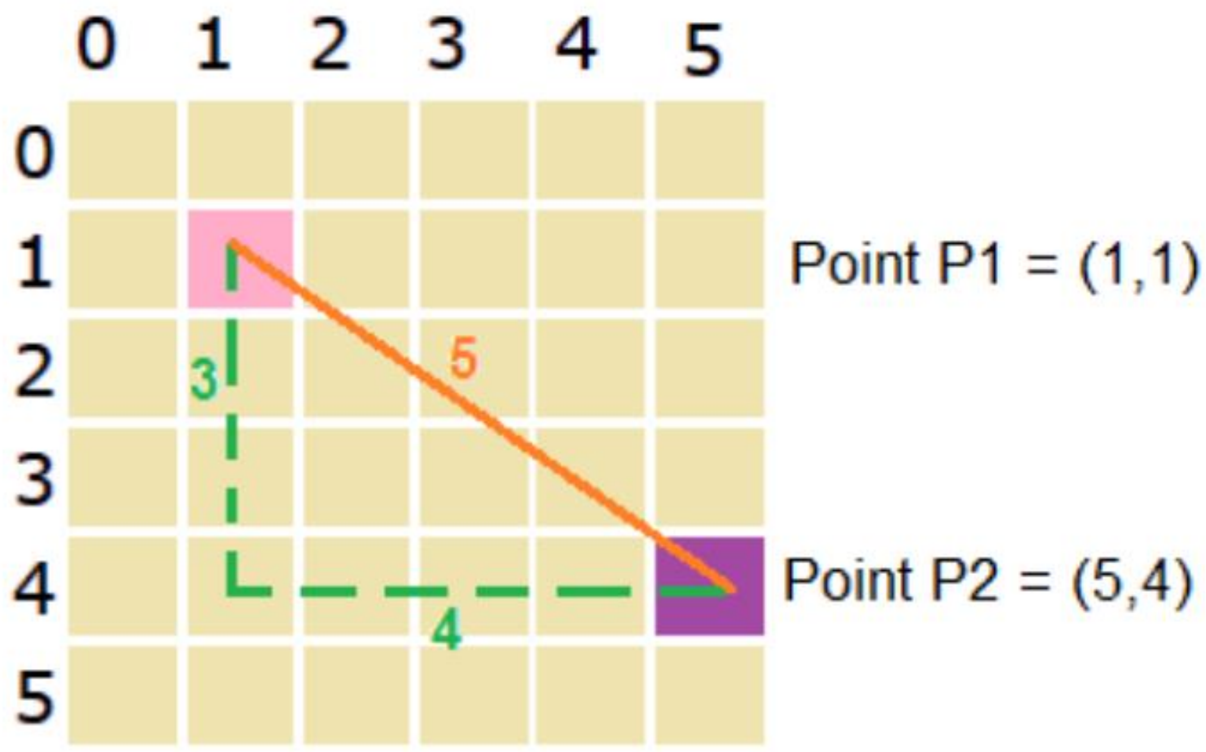
It follows grid-like paths similar to navigating city blocks.

Formula

$\Sigma|x_i - y_i|$ for all coordinates in the feature vectors.

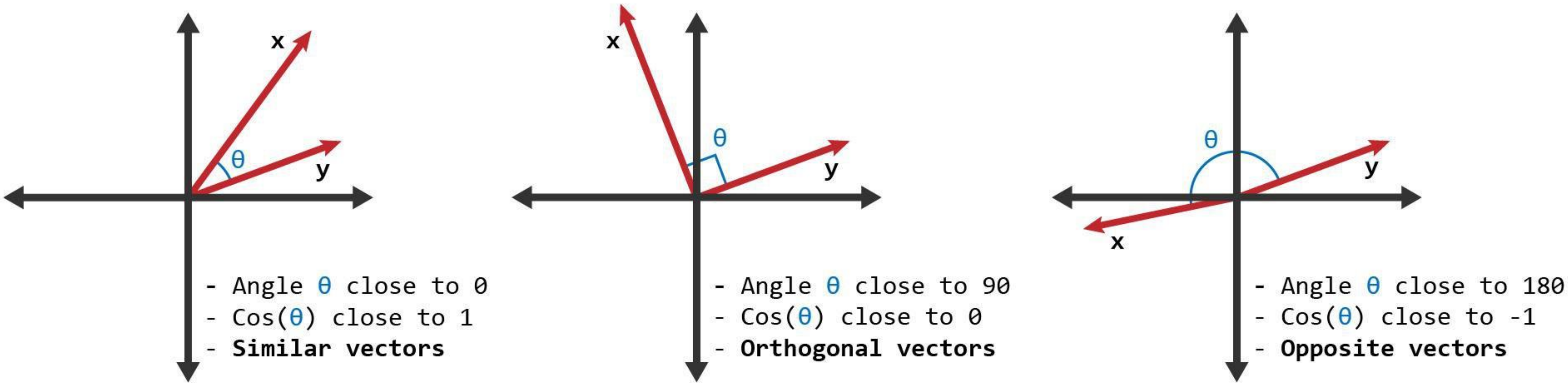


Manhattan Distance vs Euclidean Distance



Euclidean distance = $\sqrt{(5-1)^2 + (4-1)^2} = 5$

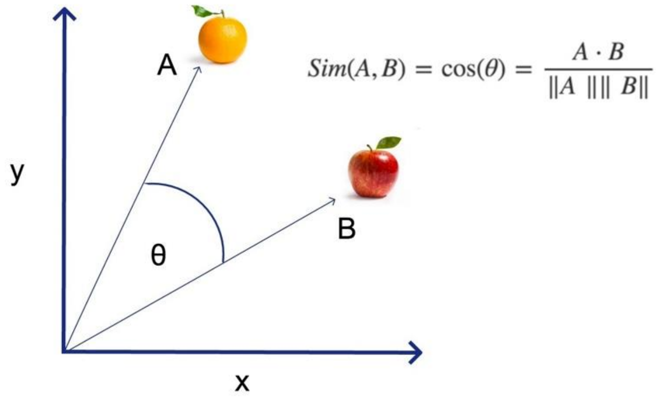
Manhattan distance = $|5-1| + |4-1| = 7$



Cosine Similarity

Angle-Based Measurement

Measures the cosine of the angle between two non-zero vectors.



Formula Application

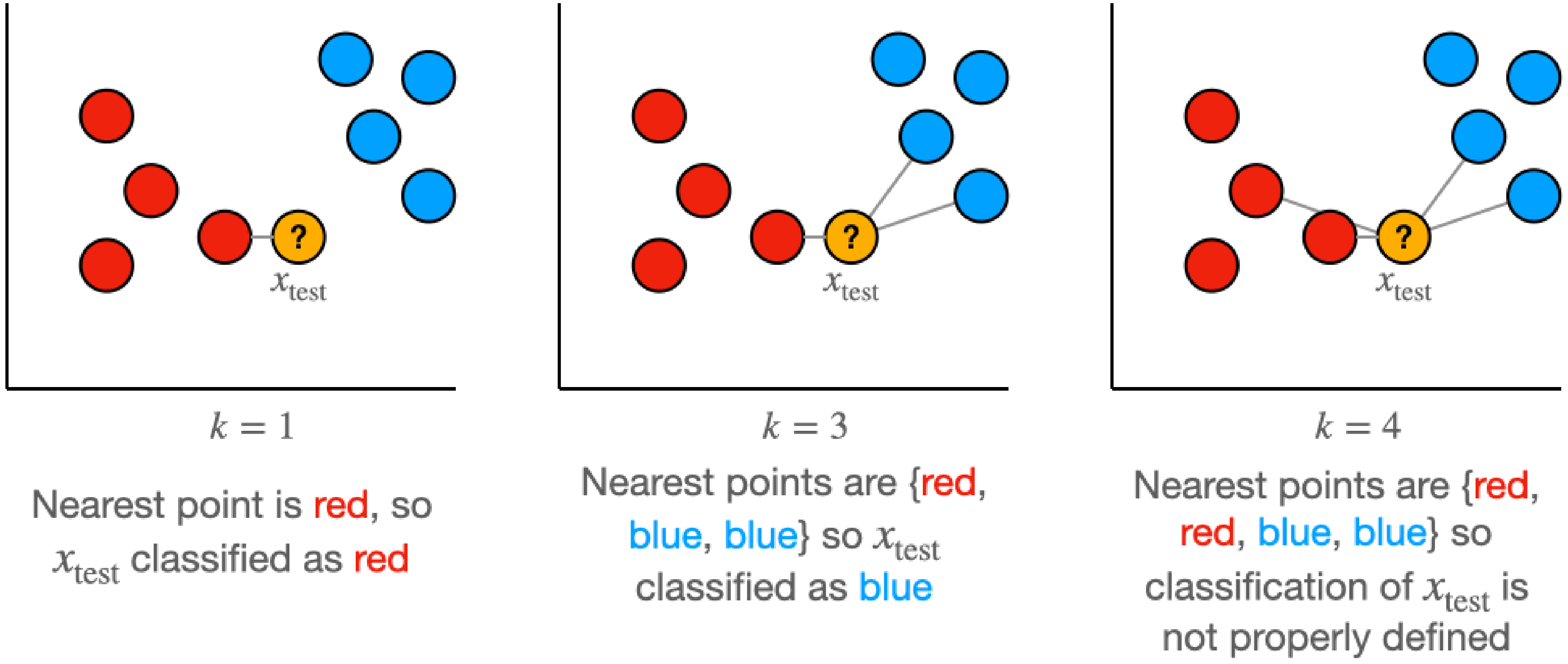
$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

High-Dimensional Data

Excels with sparse vectors like TF-IDF representations.

https://businessanalytics.substack.com/p/cosine-similarity-explained
https://www.learndatasci.com/glossary/cosine-similarity/

K-Nearest Neighbors



https://towardsdatascience.com/why-does-increasing-k-decrease-variance-in-knn-9ed6de2f5061/

K-Nearest Neighbors Overview

Classification

Predicts class based on majority vote of nearest neighbors

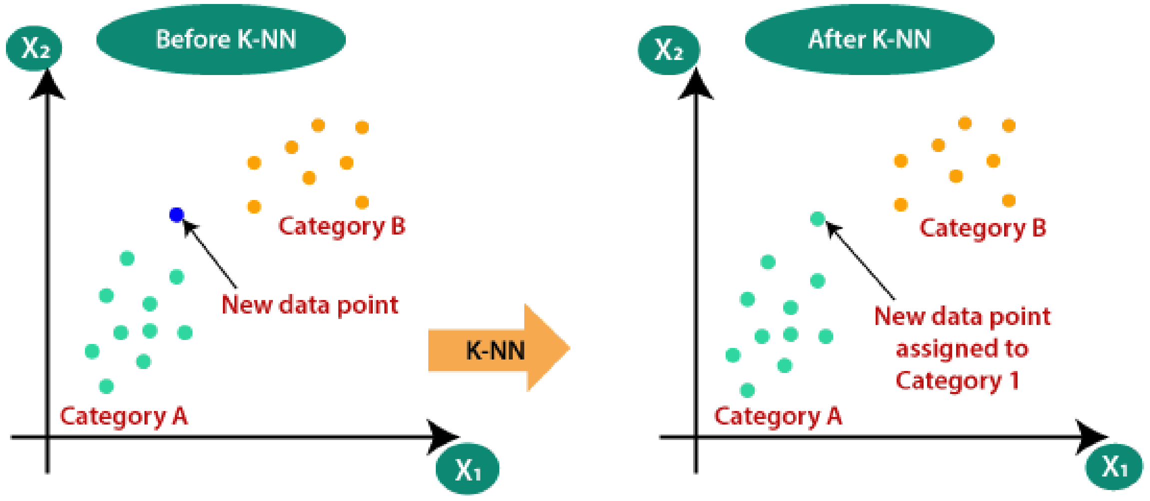
Regression

Predicts values using averages of nearest neighbors

Storage

Stores all training examples as reference points

K-NN is intuitive and versatile, requiring no training phase beyond storing examples. It's memory-intensive but highly flexible for various problem types.



K-NN Implementation Steps

Choose Value of K

Select an optimal K value through cross-validation. K should be odd to avoid ties.

Calculate Distances

Compute distances between test example and all training examples. Use Euclidean, Manhattan, or Cosine metrics.

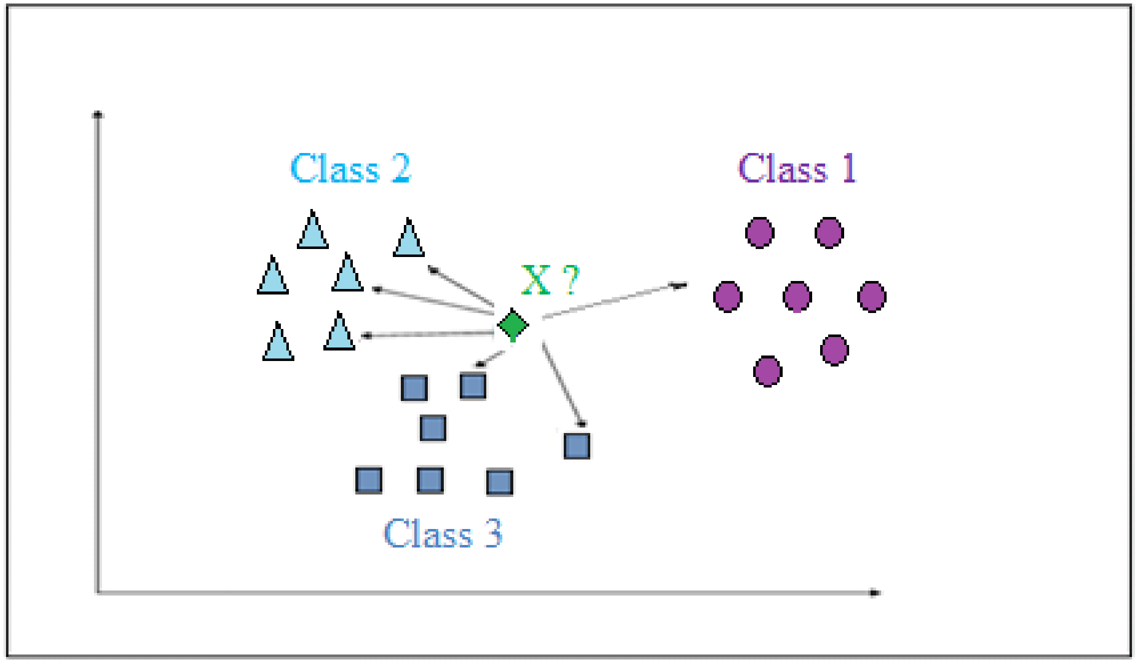
Find K Nearest Neighbors

Sort distances and identify the K closest training examples. This subset determines the classification.

Apply Voting Mechanism

Use **majority** or **weighted voting** to determine the predicted class. Return the result.

https://trendspider.com/learning-center/k-nearest-neighbors-knn/



Majority Voting in K-NN

How It Works

Each of the K nearest neighbours casts one equal vote. The class with the most votes wins.

Simple to implement and explain, requiring only vote counting.

Implementation

Count occurrences of each class among K neighbors. Return the most frequent class.

Challenges

Ties can occur with even K values. Solution: use odd K or add tiebreaker rule.

All neighbors have equal influence regardless of distance.

https://www.researchgate.net/figure/The-Majority-Voting-KNN-K6_fig3_286477914

Weighted Voting Technique

- By the **inverse linear distance** from the test instance to instance *j*

$$w_j = \frac{d_{max} - d_j}{d_{max} - d_{min}}$$

where d_{min} is for the nearest neighbour of the test instance, and d_{max} is for the furthest neighbour of the test instance

- By the **inverse distance** from the test instance to instance *j*

$$w_j = \frac{1}{d_j + \epsilon}$$

Vote Calculation

Sum weights for each class; highest weighted class wins

- What is the class label using different weighting strategies?

Instance	Class	Distance
d_1	no	0
d_2	yes	1
d_3	yes	1.5
d_4	yes	2

- **Equal weight** (majority voting):
  yes = 3
  no = 1
- **Inverse linear distance** voting:
  d_{min}= 0, d_{max}= 2,
  yes = $(\frac{1}{2} + \frac{0.5}{2} + 0) = \frac{3}{4}$
  no = 1
- **Inverse distance** voting ($\epsilon = 0.5$)
  yes = $(\frac{1}{1.5} + \frac{1}{2} + \frac{1}{2.5})$ =1.57
  no = $\frac{1}{0.5} = 2$

K-NN classification

Consider the following dataset

Training set:

APPLE	IBM	LEMON	SUN	CLASS
4	0	1	1	fruit
5	0	5	2	fruit
2	5	0	0	computer
1	2	1	7	computer

Test set:

APPLE	IBM	LEMON	SUN	CLASS
2	0	3	1	?
1	2	1	0	?

Q1

Classify the test instances using 1-NN and 3-NN with various distance measures (Euclidean distance, Manhattan distance, cosine similarity). For 3-NN, consider both majority vote and weighted voting (cosine similarity can be weighted by simply summing the similarities of the 3 neighbours). Complete the tables below. How does the classification of each test instance change with different parameters?

Euclidean distance

The Euclidean distances to the neighbors for test instance 1 are:

A. $\sqrt{(2-4)^2+(0-0)^2+(3-1)^2+(1-1)^2}=\sqrt{8}=2.828$ (class = fruit)

B. $\sqrt{(2-5)^2+(0-0)^2+(3-5)^2+(1-2)^2}=\sqrt{14}=3.742$ (class = fruit)

C. $\sqrt{(2-2)^2+(0-5)^2+(3-0)^2+(1-0)^2}=\sqrt{35}=5.916$ (class = computer)

D. $\sqrt{(2-1)^2+(0-2)^2+(3-1)^2+(1-7)^2}=\sqrt{45}=6.708$ (class = computer)

The Euclidean distances to the neighbors for test instance 2 are:

A. $\sqrt{(1-4)^2+(2-0)^2+(1-1)^2+(0-1)^2}=\sqrt{14}=3.742$ (class = fruit)

B. $\sqrt{(1-5)^2+(2-0)^2+(1-5)^2+(0-2)^2}=\sqrt{40}=6.325$ (class = fruit)

C. $\sqrt{(1-2)^2+(2-5)^2+(1-0)^2+(0-0)^2}=\sqrt{11}=3.317$ (class = computer)

D. $\sqrt{(1-1)^2+(2-2)^2+(1-1)^2+(0-7)^2}=\sqrt{49}=7$ (class = computer)

Inverse distance

fruit: $\frac{1}{2.828+\epsilon} + \frac{1}{3.742+\epsilon} = 0.621$

computer: $\frac{1}{5.916+\epsilon} = 0.169$

and the test instance 1 label is `fruit`.

Inverse linear distance

if using inverse linear distance to classify test instance 1, the weights are:

fruit: $\frac{5.916-2.828}{5.916-2.828} + \frac{5.916-3.742}{5.916-2.828} = 1.704$

computer: $\frac{5.916-5.916}{5.916-2.828} = 0$

and the test instance 1 label is `fruit`.

Manhattan distance

The Manhattan distances to the neighbors for test instance 1 are:

A. $|2-4|+|0-0|+|3-1|+|1-1|=4$ (class = fruit)

B. $|2-5|+|0-0|+|3-5|+|1-2|=6$ (class = fruit)

C. $|2-2|+|0-5|+|3-0|+|1-0|=9$ (class = computer)

D. $|2-1|+|0-2|+|3-1|+|1-7|=11$ (class = computer)

The Manhattan distances to the neighbors for test instance 2 are:

A. $|1-4|+|2-0|+|1-1|+|0-1|=6$ (class = fruit)

B. $|1-5|+|2-0|+|1-5|+|0-2|=12$ (class = fruit)

C. $|1-2|+|2-5|+|1-0|+|0-0|=5$ (class = computer)

D. $|1-1|+|2-2|+|1-1|+|0-7|=7$ (class = computer)

(cosine similarity can be weighted by simply summing the similarities of the 3 neighbours)

A value of 1 indicates perfect similarity, 0 indicates orthogonality (no similarity), -1 indicates perfect dissimilarity.

Cosine similarity

Cosine similarity between two vectors A, B is $\frac{(A \cdot B)}{\|A\| \|B\|}$

The cosine similarities to the neighbors for test instance 1 are:

A. $((2 * 4) + (0 * 0) + (3 * 1) + (1 * 1))/(\sqrt{4 + 0 + 9 + 1} * \sqrt{16 + 0 + 1 + 1}) = 0.756$ (class = fruit)

B. $((2 * 5) + (0 * 0) + (3 * 5) + (1 * 2))/(\sqrt{4 + 0 + 9 + 1} * \sqrt{25 + 0 + 25 + 4}) = 0.982$ (class = fruit)

C. $((2 * 2) + (0 * 5) + (3 * 0) + (1 * 0))/(\sqrt{4 + 0 + 9 + 1} * \sqrt{4 + 25 + 0 + 0}) = 0.199$ (class = computer)

D. $((2 * 1) + (0 * 2) + (3 * 1) + (1 * 7))/(\sqrt{4 + 0 + 9 + 1} * \sqrt{1 + 4 + 1 + 49}) = 0.432$ (class = computer)

The cosine similarities to the neighbors for test instance 2 are:

A. $((1 * 4) + (2 * 0) + (1 * 1) + (0 * 1))/(\sqrt{1 + 4 + 1 + 0} * \sqrt{16 + 0 + 1 + 1}) = 0.481$ (class = fruit)

B. $((1 * 5) + (2 * 0) + (1 * 5) + (0 * 2))/(\sqrt{1 + 4 + 1 + 0} * \sqrt{25 + 0 + 25 + 4}) = 0.556$ (class = fruit)

C. $((1 * 2) + (2 * 5) + (1 * 0) + (0 * 0))/(\sqrt{1 + 4 + 1 + 0} * \sqrt{4 + 25 + 0 + 0}) = 0.910$ (class = computer)

D. $((1 * 1) + (2 * 2) + (1 * 1) + (0 * 7))/(\sqrt{1 + 4 + 1 + 0} * \sqrt{1 + 4 + 1 + 49}) = 0.330$ (class = computer)

Test instance 1

APPLE	IBM	LEMON	SUN	CLASS
2	0	3	1	?

Measure	K	Weight	Prediction
Euclidean	1	N/A	fruit
Euclidean	3	Majority vote	fruit
Euclidean	3	Inverse dist	fruit
Euclidean	3	Inverse linear dist	fruit
Manhattan	1	N/A	fruit
Manhattan	3	Majority vote	fruit
Manhattan	3	Inverse dist	fruit
Manhattan	3	Inverse linear dist	fruit
Cosine	1	N/A	fruit
Cosine	3	Majority vote	fruit
Cosine	3	Sum	fruit

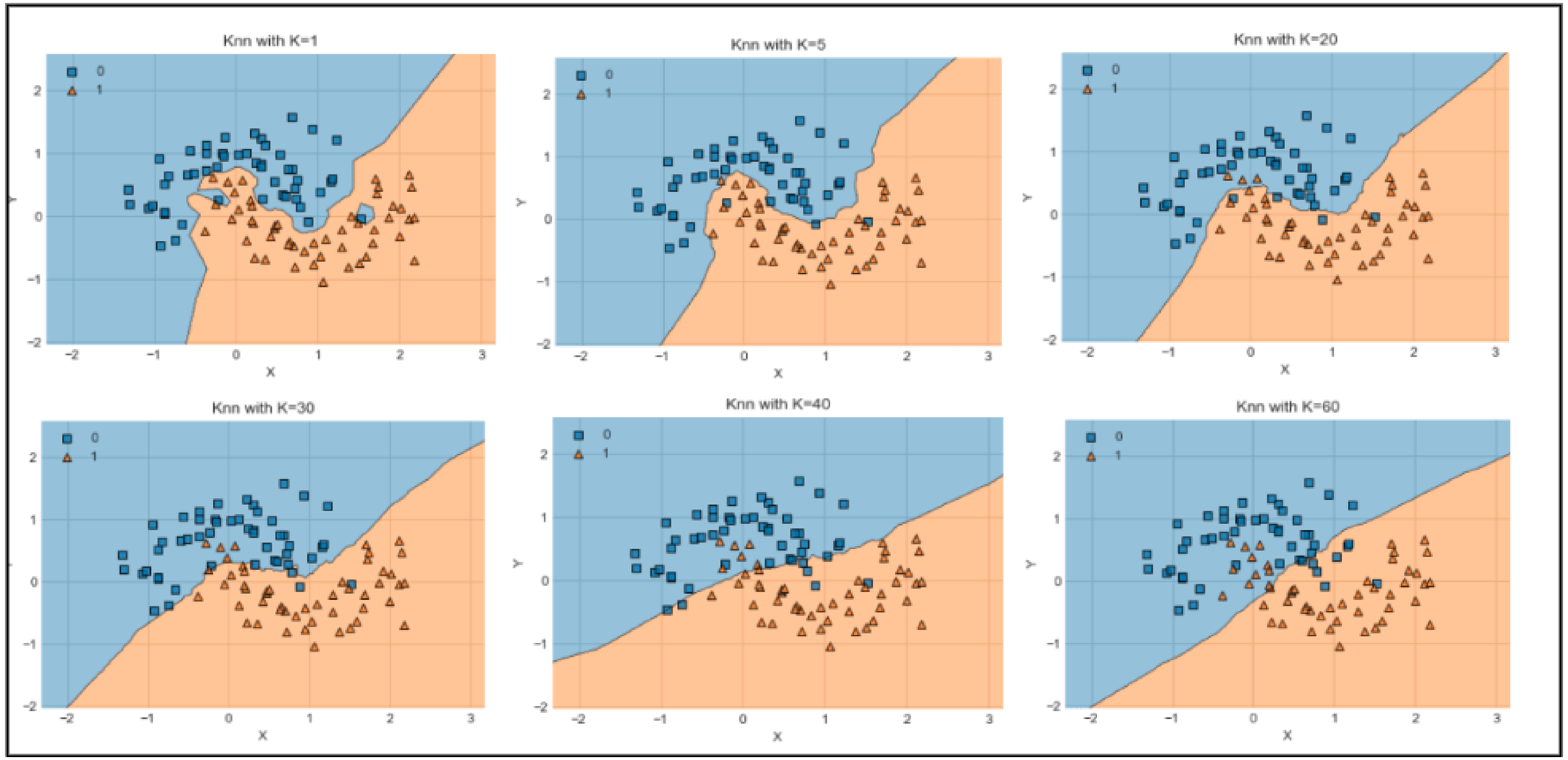
Test instance 2

APPLE	IBM	LEMON	SUN	CLASS
1	2	1	0	?

Measure	K	Weight	Prediction
Euclidean	1	N/A	computer
Euclidean	3	Majority vote	fruit
Euclidean	3	Inverse dist	fruit
Euclidean	3	Inverse linear dist	computer
Manhattan	1	N/A	computer
Manhattan	3	Majority vote	computer
Manhattan	3	Inverse dist	computer
Manhattan	3	Inverse linear dist	computer
Cosine	1	N/A	computer
Cosine	3	Majority vote	fruit
Cosine	3	Sum	fruit

Impact of K Value

K directly controls decision boundary complexity. Lower K creates irregular boundaries sensitive to local patterns. Higher K produces smoother boundaries capturing general trends.



Small K Values (K=1, 3)

Creates complex, highly flexible decision boundaries.

Fits training data closely but risks overfitting to noise.

Large K Values (K=10+)

Creates smoother, more generalized boundaries.

If the dataset has high class imbalance, setting k to a high value may make the classifier unable to recognize the rare classes.

Finding Optimal K

Use cross-validation to test different K values.

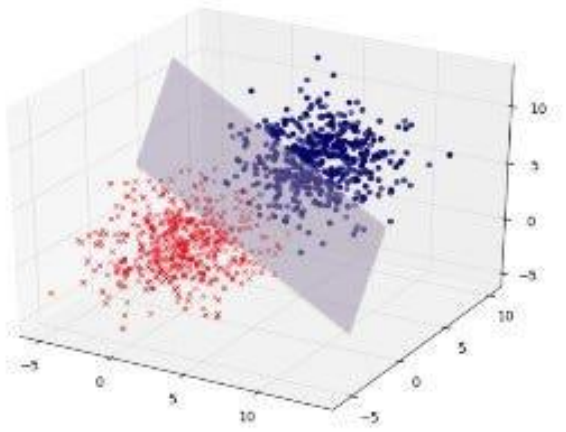
https://neptune.ai/blog/knn-algorithm

Support Vector Machines Introduction

SVM tries to **find the best boundary (hyperplane)** that separates data points of different classes.

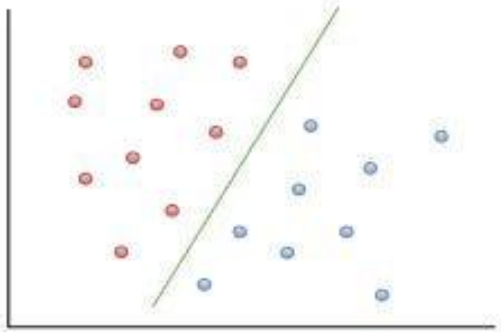
$$\mathbf{w}^T\mathbf{x} = 0$$

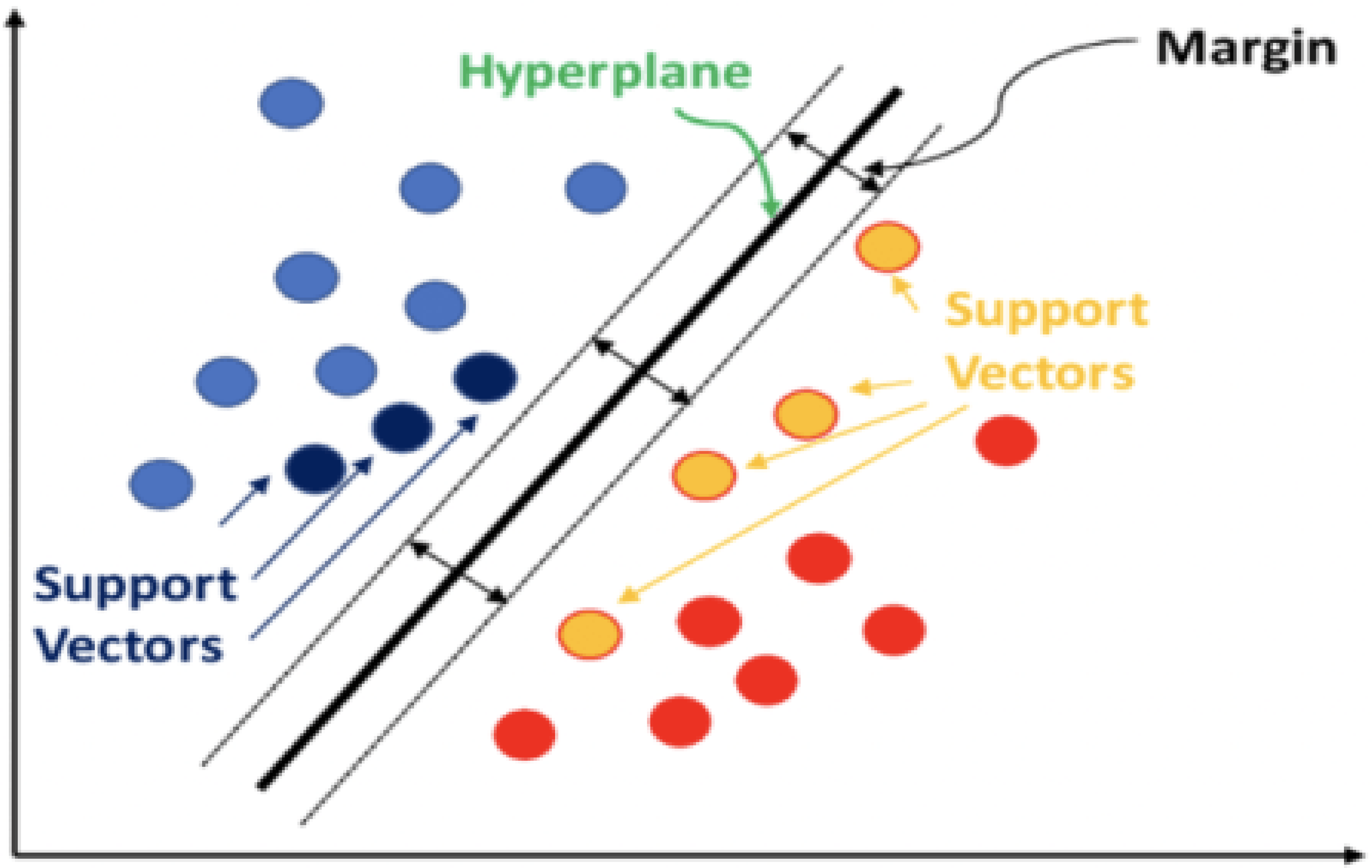
Hyperplane



$$y = ax + b$$

Line





•**Hyperplane** – *Hyperplane is the decision boundary that aids in classifying the data points.*

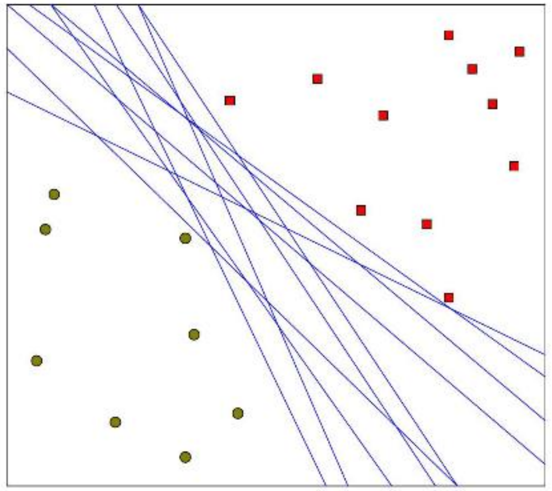
•**Support Vectors** – *Support Vectors are the data points that are on or nearest to the hyperplane and influence the position of the hyperplane.*

•**Margin –** *Margin is the gap between the hyperplane and the support vectors.*

•**Kernel function** – *These are the functions used to determine the shape of the hyperplane. Transforms data to handle non-linear classification.*

https://datatron.com/what-is-a-support-vector-machine/

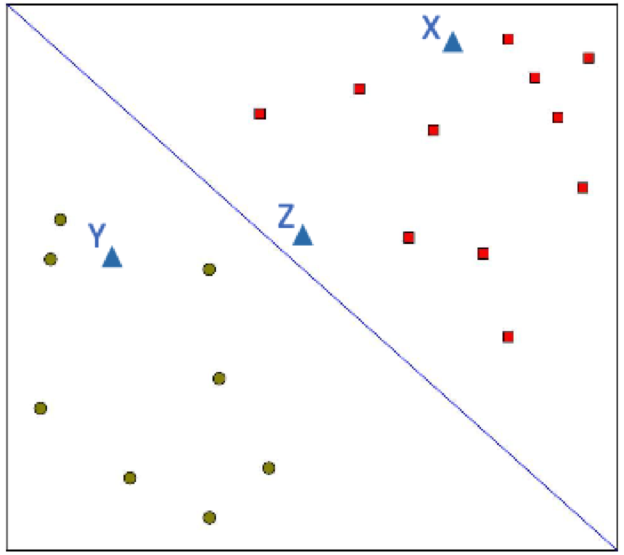
Support Vector Machines Introduction



Hyperplane Optimization

Finds optimal decision boundary with maximum margin

- Consider the distance from a data point to the boundary

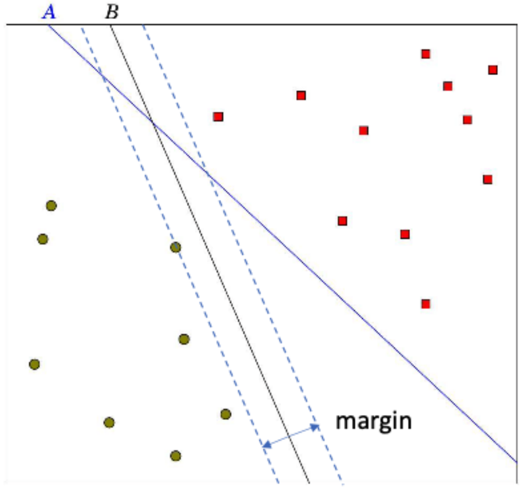


- For point X, we should be quite confident about the prediction of its class.
- For point Z, a small change to the decision boundary might change our decision to change; we are less confident in the prediction.

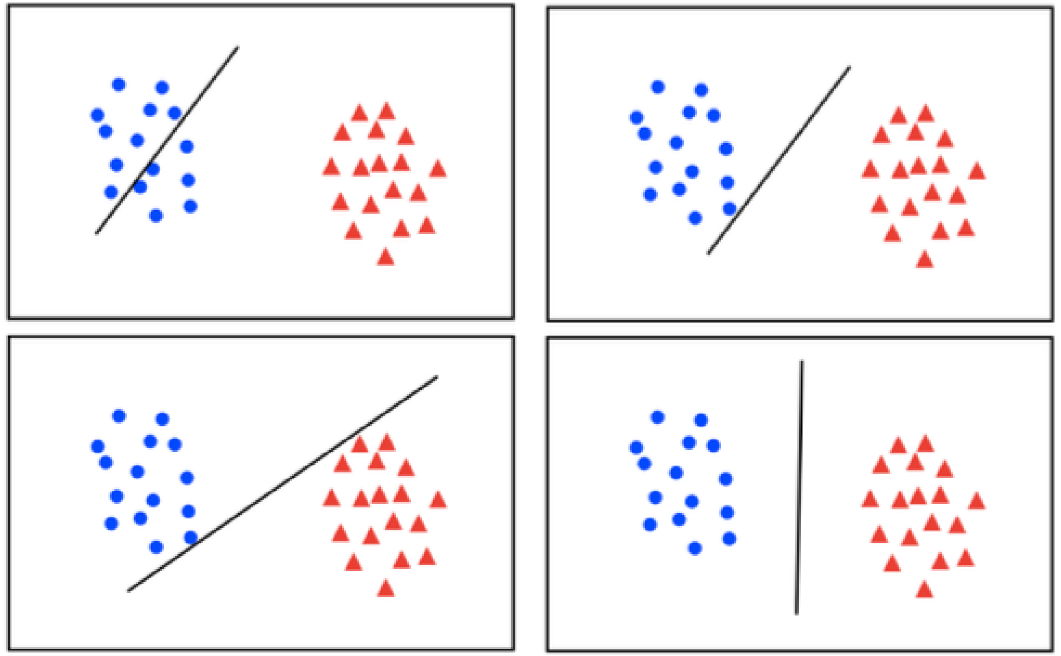
Hyperplane Optimization

- Aim: find a decision boundary that allows to make all correct and confident (far from the decision boundary) predictions for a given training set.
- SVM finds an optimal solution
  - *Maximises the distance* between the hyperplane and the difficult points close to decision boundary

Margin: 2 x the minimum distance between boundary and data points



- What is the best hyperplane?



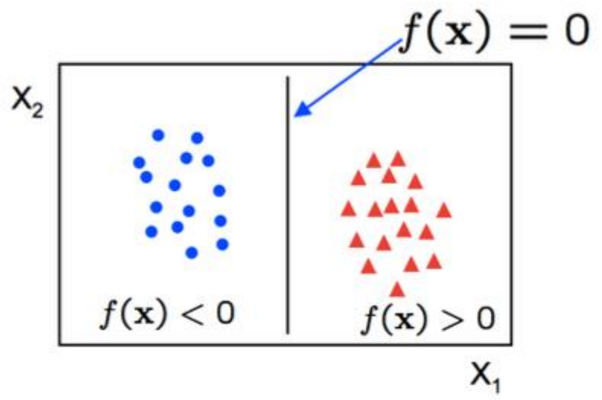
Linear Classifier

- A separating hyperplane in D dimensions can be defined by a **normal $\boldsymbol{w}$** and an **intercept** b
- The hyperplane passing a point $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \ldots \\ x_D \end{bmatrix}$ is:

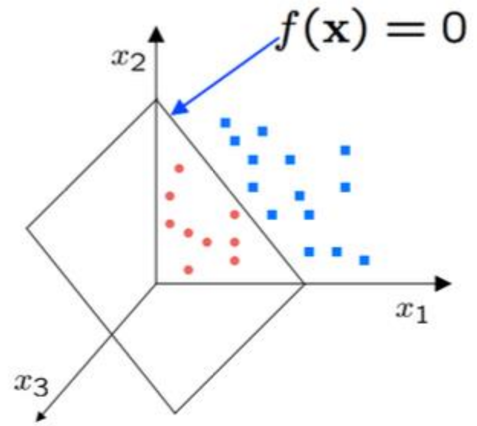
$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = 0 \qquad \boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \\ \ldots \\ w_D \end{bmatrix}$$

$$w_1 x_1 + \ldots + w_D\, x_D + b = 0$$

- Linear classifier takes the form $f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b$
- In 2D space, this is a straight line



- Linear classifier takes the form $f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b$
- In 3D space, this is a plane



Linearly Separable Classification

Perfect Separation

Classes can be completely divided by a linear hyperplane.

Maximum Margin

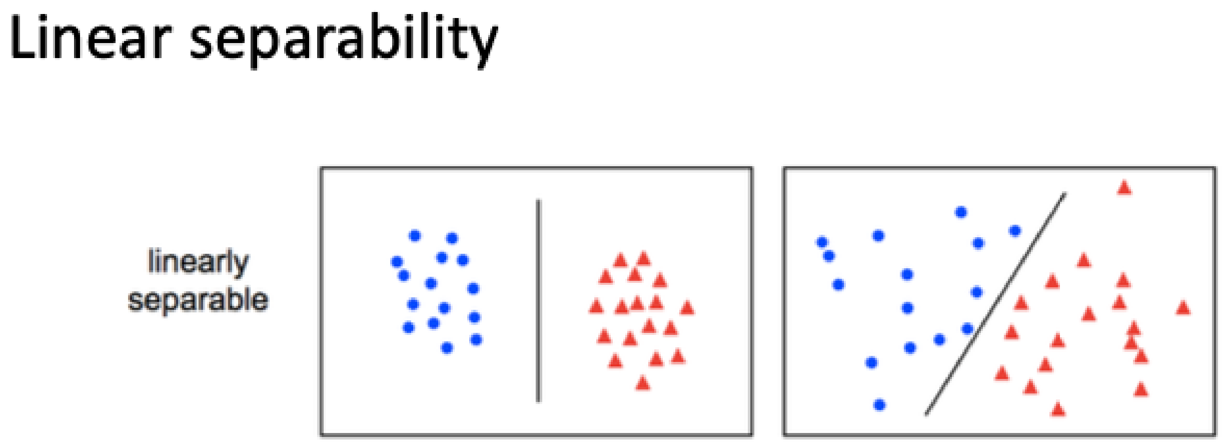
SVM finds the widest possible gap between classes.

Support Vectors

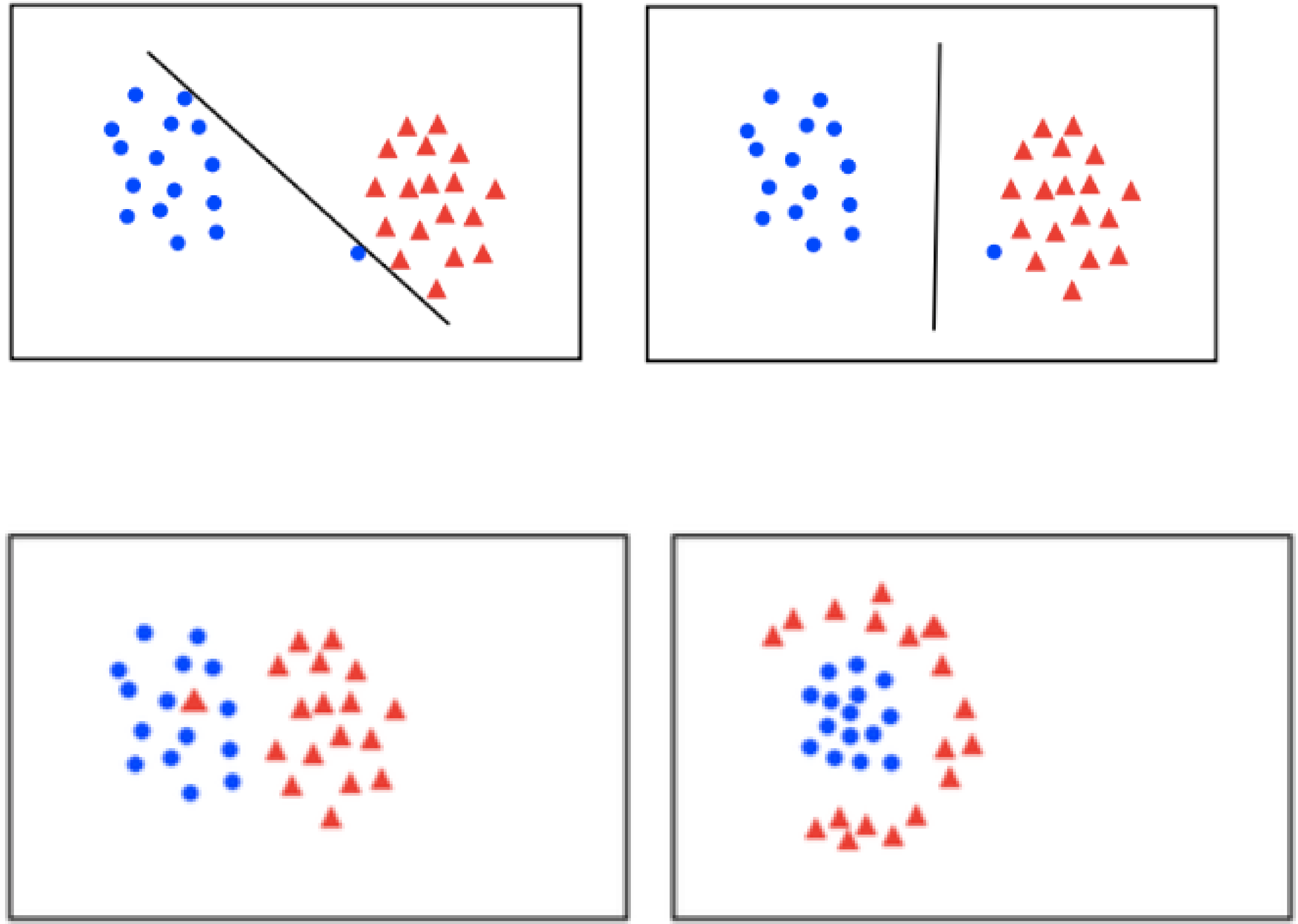
Only points nearest to boundary influence the result.

Robustness

Maximum margin provides better generalization.

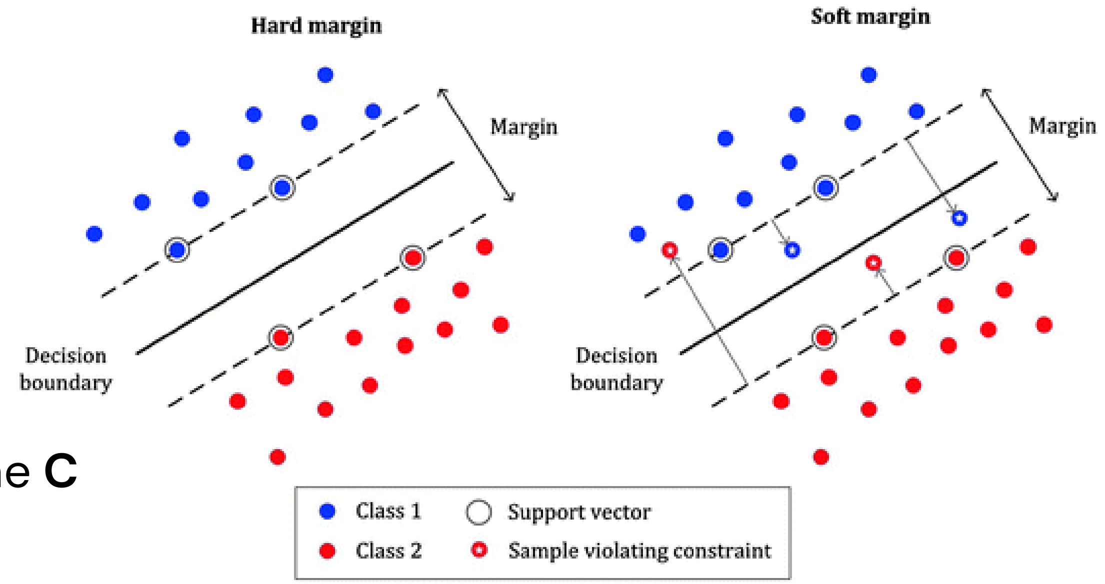


But What if Data is Not Linearly Separable?



not linearly separable

Two Solutions



1

Allow Soft Margin (C Parameter)

•Introduce a **soft margin** to allow some **misclassifications** using the **C parameter**.

•This lets the SVM **tolerate overlap** between classes while still finding the best possible boundary.

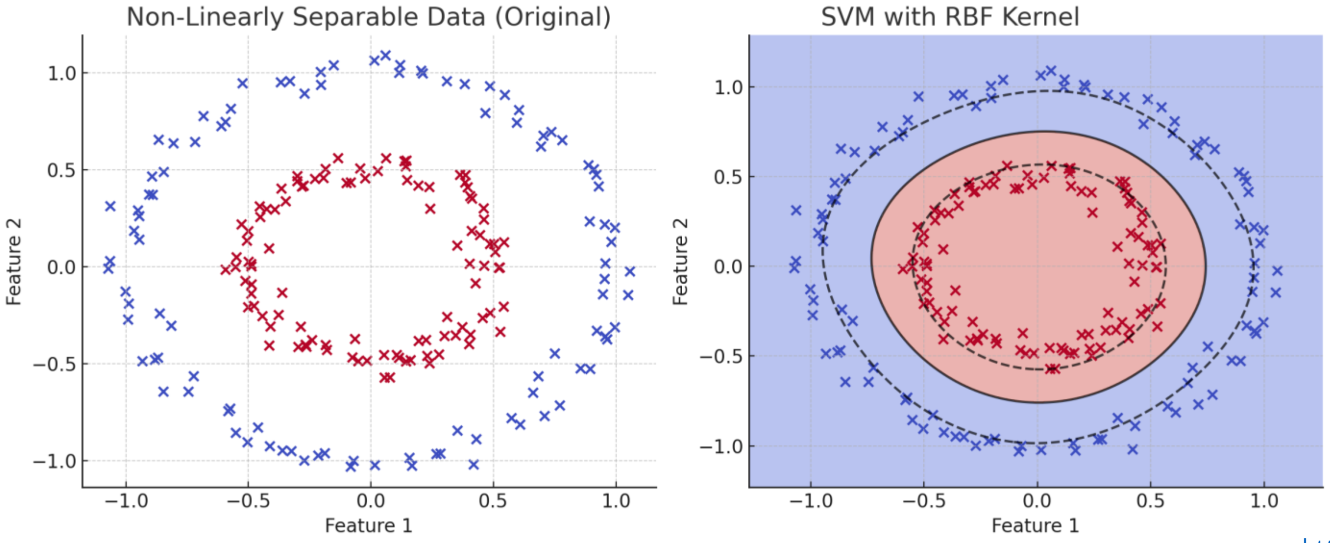
2

Use the Kernel Trick

•Apply a **non-linear kernel function** to **map the data into a higher-dimensional space** where it becomes linearly separable.

•Common kernels:

- **RBF (Gaussian)**
- **Polynomial**
- **Sigmoid**



https://doi.org/10.1142/S1469026808002314

Soft margins

Soft Margins

- Introduce *slack variables* $\{\xi_1, \xi_2, \ldots, \xi_i, \ldots, \xi_N\}$, which allows few points to be on the "wrong" side of the hyperplane at some cost
- New objective function with slack variables

$$\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\xi_i$$

Slack Penalty

Slack Variable

"The extent to which a data point violates the margin or is misclassified"

"How much should we **penalize violations** of the margin?"

In real-world datasets, classes are often **not perfectly separable**. So, SVM allows some points to:

•Be **on the wrong side of the margin** (i.e., lie **within the margin boundaries** as in Fig. B)

•Even be **misclassified (as in Fig. A)**

Using **Slack variables (ξ)**

Fig. A

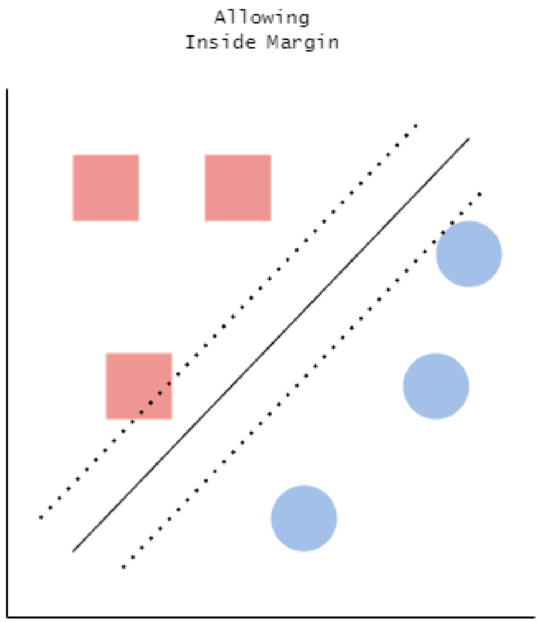


Fig. B

https://helenedk.medium.com/an-introduction-to-soft-margin-support-vector-machines-fec8420981af

Soft Margins

- Introduce *slack variables* $\{\xi_1, \xi_2, \ldots, \xi_i, \ldots, \xi_N\}$, which allows few points to be on the "wrong" side of the hyperplane at some cost
- New objective function with slack variables

$$\min_{w} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N} \xi_i$$

•Here, w is the normal vector to the hyperplane
•||w|| controls the margin → **smaller w → wider margin**
•SVM **minimizes the norm of w** to **maximize the margin**

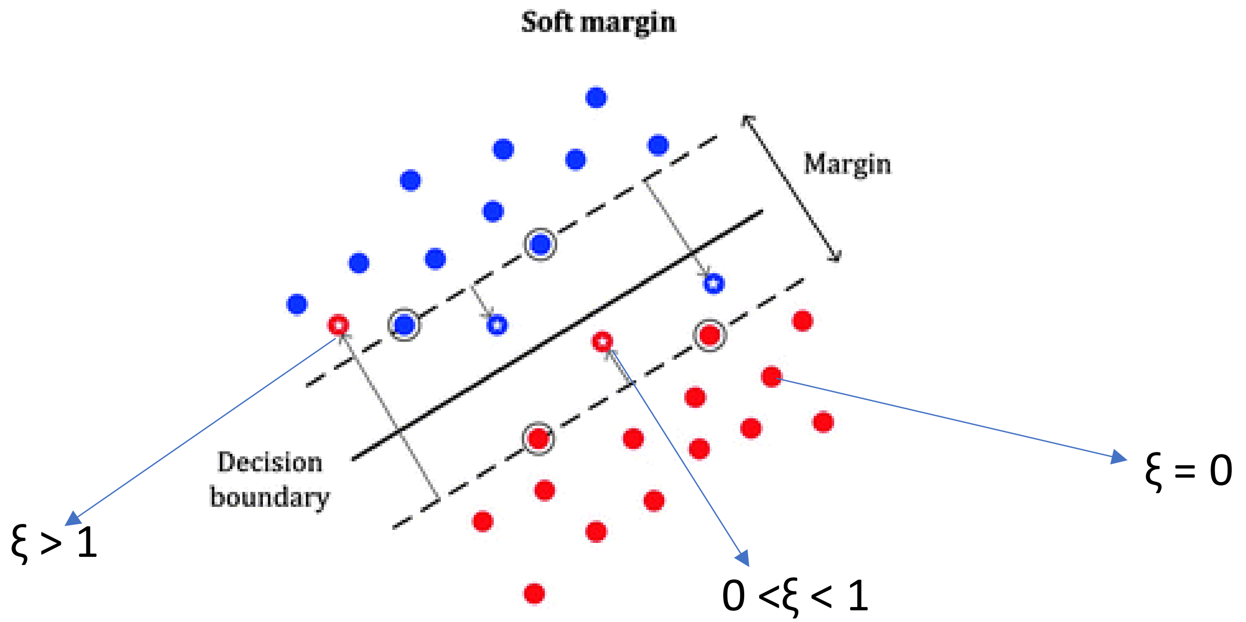
C is a **regularization parameter**:
•**Low C** → wider margin, more misclassifications
•**High C** → narrow margin, fewer violations

Slack Variables in SVM

🔷 What is a **Slack Variable** (ξ_i)?

For each data point `i`, slack variable `ξᵢ` tells **how much it violates** the margin:

Case	**ξ_i value**
Correctly classified and outside margin	$\xi_i = 0$
Inside the margin but on correct side	$0 < \xi_i < 1$
Misclassified point	$\xi_i > 1$



Slack Penalty

C=0.1

Low Penalty

Allows many violations, creating smoother boundaries

C=1.0

Balanced Penalty

Moderate trade-off between errors and margin width

C=10

High Penalty

Strict boundaries with fewer misclassifications allowed

C=100

Extreme Penalty

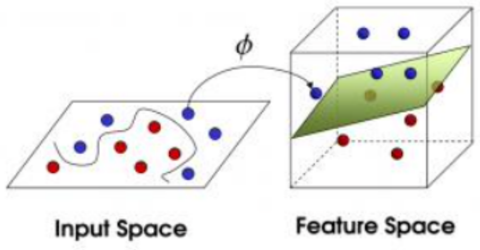
Nearly rigid separation, risking overfitting

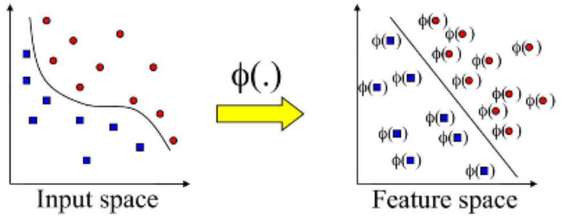
Large C → hard margin behaviour
Small C → soft margin behaviour (more forgiving)

- A **hard margin** requires perfect separation (no misclassified points).
- A **soft margin** allows:
  - Some **points to lie inside the margin**.
  - Some **points to be misclassified**.

Kernal Methods

- Make non-linearly separable problem separable
- Map data into better representation space



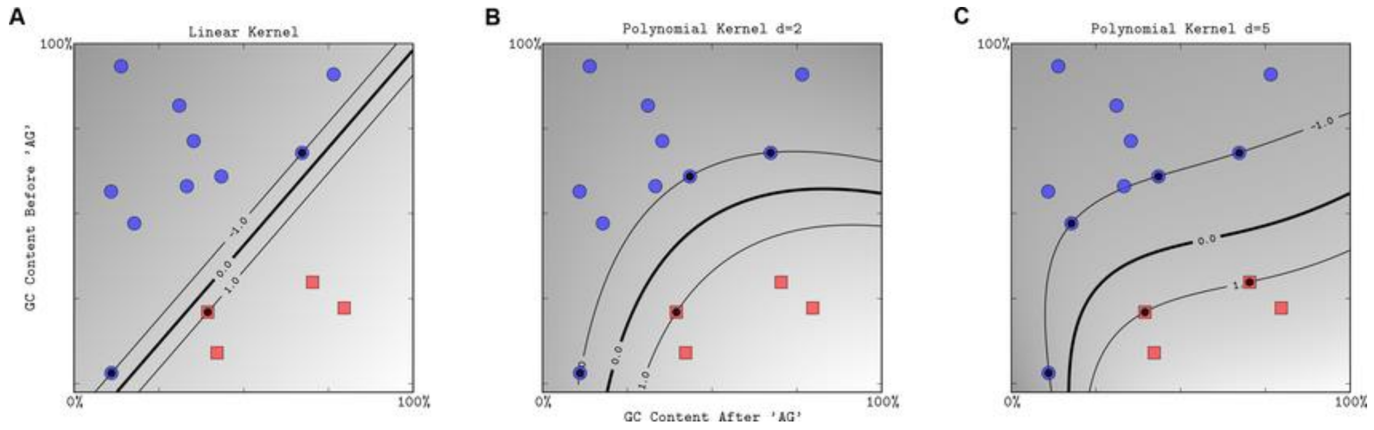


Transform the input data into a higher-dimensional space using kernel functions

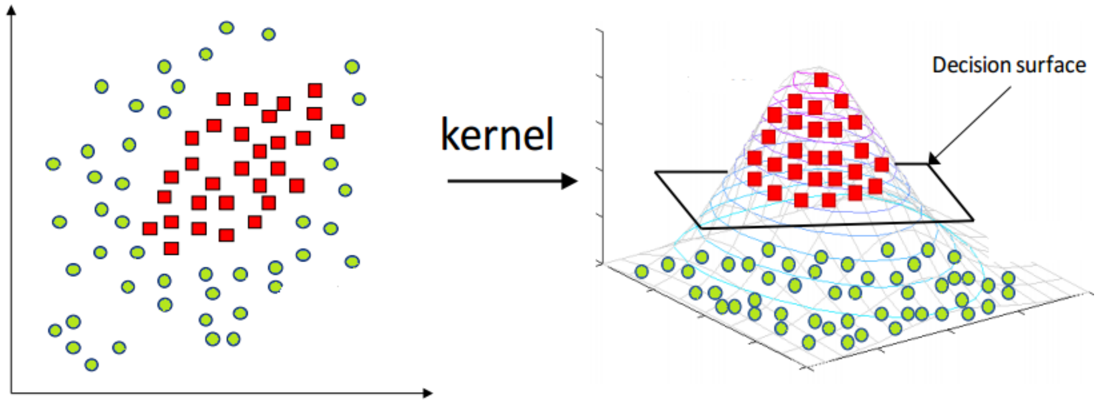
•Common kernels:

- **RBF (Gaussian)**: For circular or complex boundaries.
- **Polynomial**: Captures curved decision boundaries.
- **Sigmoid**: Similar to neural networks.

Polynomial Kernel



RBF kernel



https://itsudit.medium.com/svm-kernels-understanding-the-role-of-kernels-in-support-vector-machines-6bf1e702737a

https://medium.com/@abhishekjainindore24/svm-kernels-and-its-type-dfc3d5f2dcd8

https://blog.devgenius.io/machine-learning-algorithm-series-polynomial-kernel-svm-understanding-the-basics-and-applications-89b4b42df137

Q3

What does it mean for a classification dataset to be "linearly separable"? If a dataset isn't linearly separable, an SVM learner has two major options. What are they, and why might we prefer one to the other?

If a dataset is "linearly separable," it is possible to completely separate the classes with a single hyperplane

If a dataset isn't linearly separable, the options for SVM are:

1. ***Soft margins***

2. ***Kernel methods***

Method	Purpose	Handles...
Soft Margin	Allow misclassifications by maximizing margin	Noisy or almost separable data
Kernel Trick	Transform data to separable space	Complex non-linear boundaries

Q6

What is the value of slack variables for data points that are correctly classified in SVMs? What should the slack penalty C be to make a soft-margin SVM function as a hard-margin SVM?

- For data points that are correctly classified, the value of their slack variable is 0
- Hard SVM don't allow any mistakes or in other words the penalty rate for the mistakes is equal to infinity (∞). To make soft SVM to act like hard SVM the slack penalty C should be very large.

Q5

How do changes in data points affect the decision boundary of an SVM?

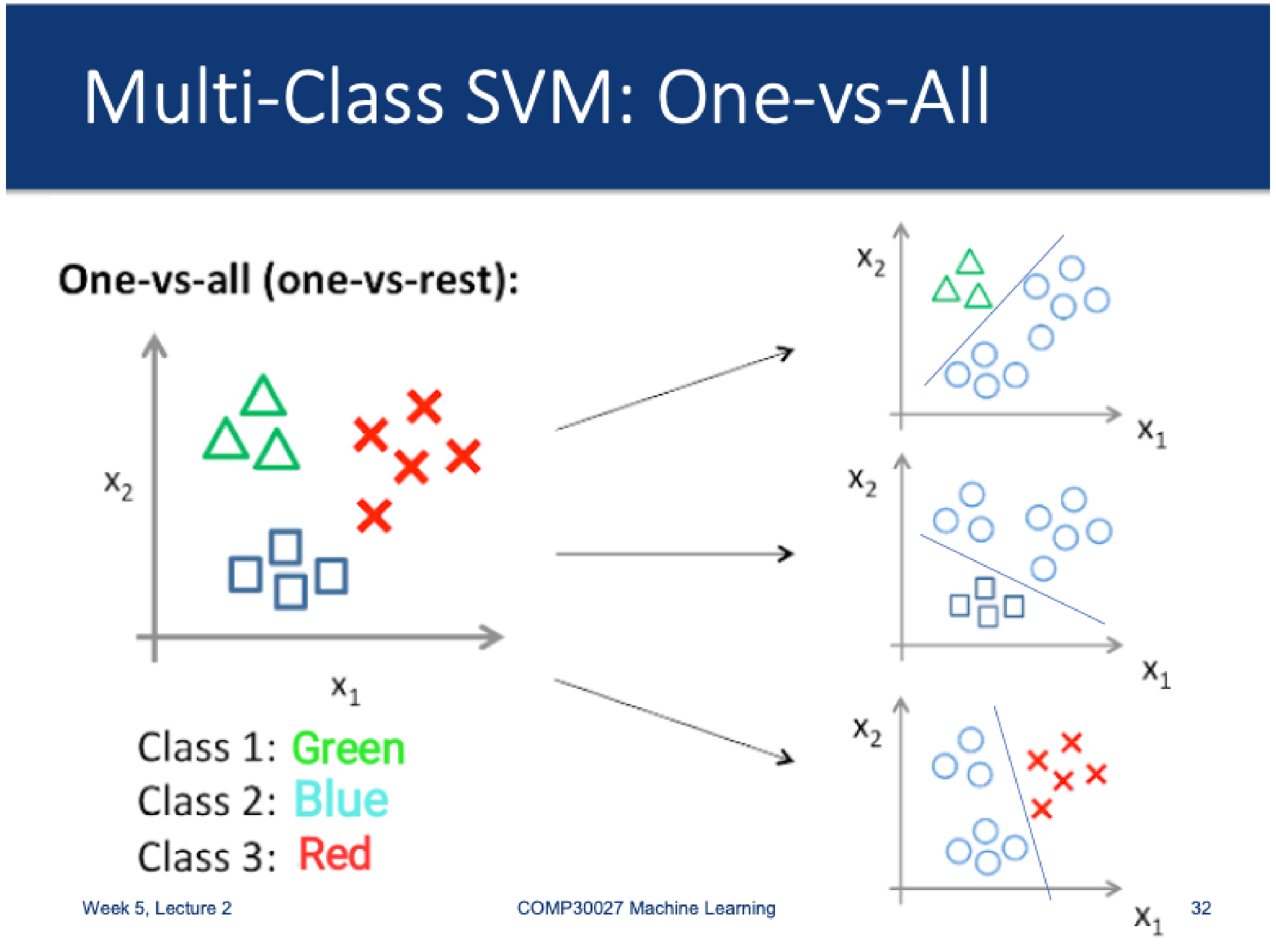
The support vectors are used to define the hyperplane that separates the classes. Changes in data points that are not support vectors have no effect on the decision boundary.

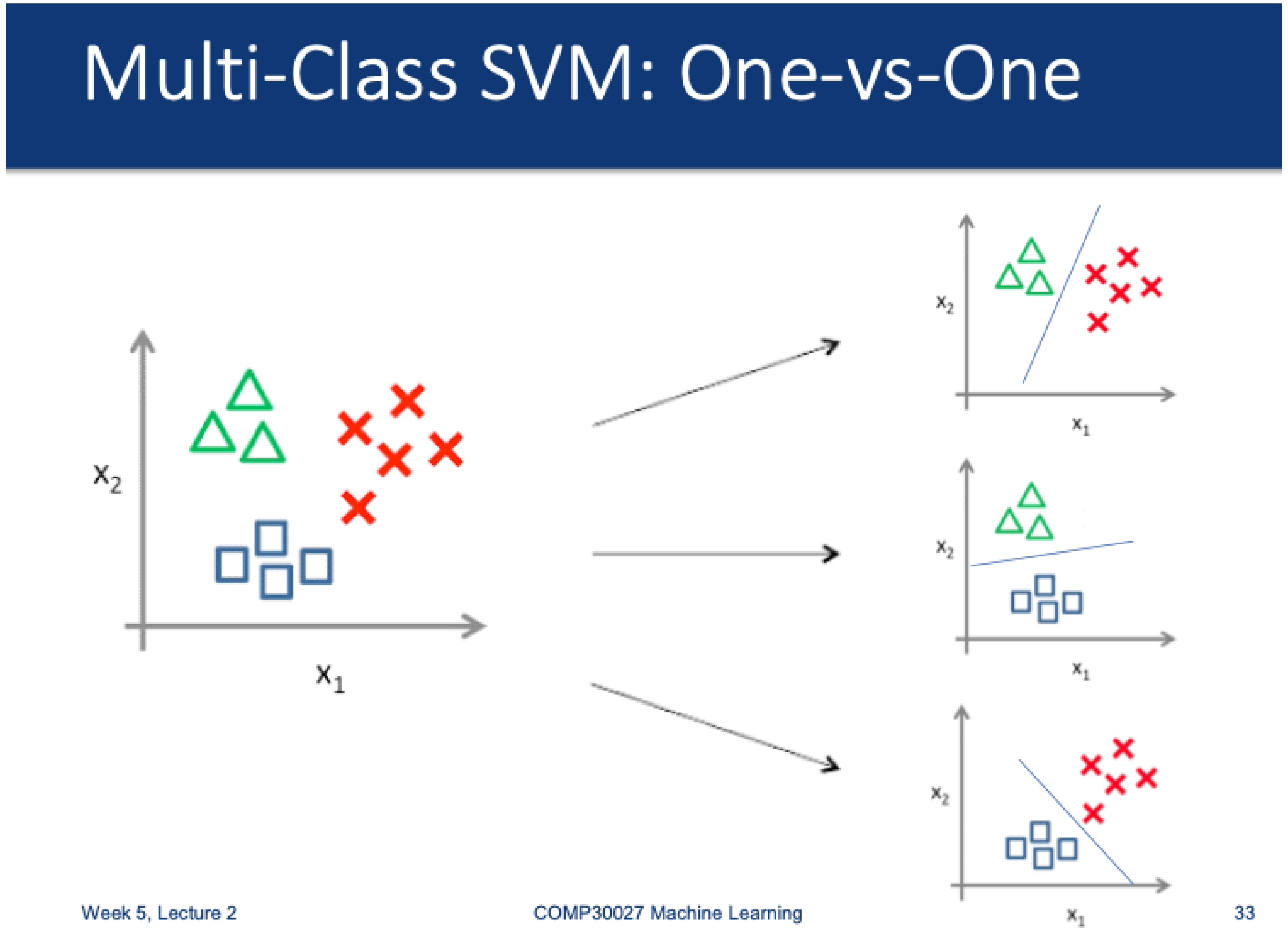
However, changes in the position or label of support vectors can have a significant impact on the decision boundary.

https://greitemann.dev/svm-demo

Multi-Class SVM

SVM is naturally binary, but we can use:





Q7

How many binary classifiers are needed to classify a dataset with 4 classes using one-vs-one method?

In the **one-vs-one (OvO)** strategy:

> **You train one binary classifier for every pair of classes.**

So, for **C** classes, the number of binary classifiers needed is:

$$\text{Number of classifiers} = \frac{C(C-1)}{2}$$

For 4 classes:

$$\frac{4(4-1)}{2} = \frac{4 \times 3}{2} = 6$$

Each classifier is trained to distinguish between **just two classes:**

- Class 0 vs 1
- Class 0 vs 2
- Class 0 vs 3
- Class 1 vs 2
- Class 1 vs 3
- Class 2 vs 3

Then during prediction:

- Each classifier votes for a class
- The class with the **most votes wins** (majority voting)

Advantages of SVMs

High-Dimensional Space

SVMs maintain effectiveness with many features. They avoid the curse of dimensionality.

The algorithm focuses on support vectors rather than all data points.

Generalization Power

Maximum margin principle provides better theoretical guarantees. SVMs often outperform K-NN.

They're less prone to overfitting with proper regularization.

Memory Efficiency

Only support vectors are stored after training. This makes prediction more efficient.

K-NN must store the entire training dataset.

Q4

Unlike other geometric methods such as K-NN, SVMs work better with large attribute sets. Why might this be true?

1. SVMs focus on the most important points (support vectors)

- SVM builds a decision boundary using only the **critical support vectors**, not all the data.
- In high dimensions, many features may be **irrelevant or noisy**, and SVMs **ignore non-support points** during decision making.

2. No need for prior feature selection; SVMs can implicitly capture the relationships between different features and can handle interactions between them without the need for explicit feature engineering. This means that SVMs can effectively learn from a large number of features without requiring a priori knowledge about which features are most important for the classification task.