Assignment Title: Image Segmentation and Image Object Classification

Course Code: CSE 405

Course Title: Digital Image Processing

Submitted by

Shanjida Alam(ID: 353)

Submitted to

Dr. Morium Akter, Professor

Computer Science and Engineering
Jahangirnagar University
Dhaka, Bangladesh

November 05, 2024

Contents

Chapter 1

Canny Edge Detector

The Canny Edge Detector is a widely-used edge detection technique in computer vision and image processing. It is designed to detect edges in an image by finding areas with significant intensity changes. The algorithm provides robust edge detection with low error rates, making it ideal for applications where accurate and well-defined edges are essential, such as object detection, image segmentation, and facial recognition.

1.1 Fundamentals of Canny Edge Detection

Edge detection involves identifying points in an image where there is a sharp change in intensity. The Canny Edge Detector achieves this by first reducing noise, then calculating intensity gradients, and finally selecting edges based on thresholds. These steps help capture important structural information within an image while minimizing the detection of irrelevant or false edges.

1.2 Steps in the Canny Edge Detection Process

The Canny Edge Detector consists of several stages, each contributing to its accurate edge detection capabilities:

1.2.1 Noise Reduction

To reduce the impact of noise, which can interfere with edge detection, the image is first smoothed using a Gaussian filter. This step reduces minor intensity fluctuations, ensuring that only significant edges are detected.

1.2.2 Gradient Calculation

After noise reduction, the intensity gradient of the image is calculated. This involves using derivative filters, such as the Sobel operator, to approximate the gradient in the x and y directions. The gradient's magnitude and direction are computed, with regions of high gradient magnitudes representing potential edges.

1.2.3 Non-Maximum Suppression

Non-maximum suppression is applied to thin out the edges. This step ensures that only the strongest points along an edge are retained by suppressing all non-maximum values along the gradient direction. This refinement results in well-defined, single-pixel-wide edges.

1.2.4 Double Thresholding

Double thresholding classifies edges based on their gradient intensity values. Two threshold values are set: a high threshold and a low threshold. Pixels with intensities above the high threshold are considered strong edges, while those between the high and low thresholds are considered weak edges. Pixels below the low threshold are discarded.

1.2.5 Edge Tracking by Hysteresis

In this final step, weak edges that are connected to strong edges are preserved, while isolated weak edges are removed. This step ensures that only meaningful edges are retained, resulting in a clean and continuous edge map.

1.3 Mathematical Representation of Gradient Calculation

The intensity gradient in the x and y directions (G_x and G_y) can be calculated using convolution with Sobel filters:

$$G_x = I * S_x \quad and \quad G_y = I * S_y$$

where I is the image and S_x and S_y are the Sobel filters in the x and y directions. The gradient magnitude G and direction θ are then given by:

$$G = \sqrt{G_x^2 + G_y^2} \quad and \quad \theta = \arctan\left(\frac{G_y}{G_x}\right)$$

1.4 Algorithm Pseudocode

A general pseudocode for the Canny Edge Detection process is as follows:

1. **Apply Gaussian filter** to the input image to reduce noise.
2. **Calculate image gradients** G x and G y using a Sobel or similar operator, then compute gradient magnitude G and direction .
3. **Apply Non-Maximum Suppression** to thin out the edges.
4. **Perform double thresholding** to classify strong and weak edges based on the gradient magnitude.
5. **Track edges by hysteresis**, connecting weak edges to strong edges where appropriate.

1.5 Diagram of Canny Edge Detection Process

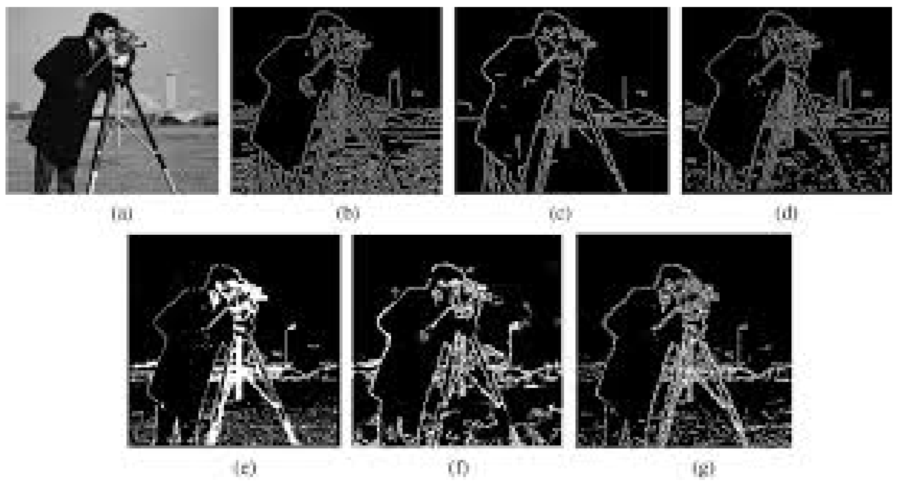

Figure 1.1: Overview of the Canny Edge Detection process, illustrating steps from noise reduction to edge tracking by hysteresis.

1.6 Applications of Canny Edge Detection

The Canny Edge Detector is utilized across various applications in image processing and computer vision:

- **Object Detection**: Canny edge maps are commonly used in object detection algorithms to locate and identify distinct objects within an image.

1.7 Applications of Canny Edge Detection

The Canny Edge Detector is widely used in various fields within image processing and computer vision. Its ability to detect edges accurately and reliably makes it valuable for multiple applications:

- **Object Detection**: The Canny edge maps are commonly used to locate and identify distinct objects in an image by highlighting their boundaries.
- **Image Segmentation**: Edge detection plays a crucial role in segmenting images into different regions by identifying the borders of various elements.
- **Facial Recognition**: The Canny Edge Detector helps identify key facial features by detecting prominent edges, such as around the eyes, nose, and mouth.
- **Medical Imaging**: In medical imaging, the Canny Edge Detector is utilized to identify and highlight outlines of organs and structures in scans, assisting in diagnostics and analysis.
- **Autonomous Vehicles**: The Canny Edge Detector helps in recognizing lane boundaries, signs, and objects on roads, aiding navigation and safety for autonomous vehicles.

Chapter 2

Corner Detection

Corner Detection is a fundamental technique in computer vision and image processing, used to identify points in an image where there is a significant change in direction. Corners, or interest points, often correspond to distinctive features in an image, such as junctions or intersections of edges. These points are highly informative for understanding the geometry and structure of objects within an image, making corner detection a critical step in applications like image matching, object tracking, and 3D reconstruction.

2.1 Fundamentals of Corner Detection

A corner can be defined as a point in an image where the direction of intensity change is different in multiple directions. Mathematically, this corresponds to regions with high gradients in both the x and y directions. Identifying corners involves detecting these changes in intensity gradients and pinpointing locations where these variations meet certain criteria.

2.2 Methods of Corner Detection

Several algorithms have been developed for corner detection, each with its unique approach to identifying corners based on intensity changes. Some of the most commonly used methods are:

2.2.1 Harris Corner Detection

The Harris Corner Detection algorithm, introduced by Chris Harris and Mike Stephens in 1988, is one of the most popular methods for detecting corners. It uses the concept of auto-correlation to measure the changes in intensity within a small window around each pixel. Harris corners are identified by evaluating the eigenvalues of the covariance matrix of image gradients.

The Harris response function is given by:

$$R = det(M) - k \cdot (trace(M))^2$$

where M is the covariance matrix of gradients in the x and y directions, and k is a constant parameter (typically between 0.04 and 0.06) that balances sensitivity. A large positive R value indicates a corner, while a negative R indicates an edge, and a small value represents a flat region.

2.2.2 Shi-Tomasi Corner Detection

The Shi-Tomasi method, an improvement on Harris Corner Detection, introduced the concept of ”good features to track.” This approach selects points where the minimum eigenvalue of the covariance matrix M is above a certain threshold. Instead of using the Harris response function, it evaluates the smaller of the two eigenvalues (λ_1 and λ_2) and selects corners based on the following criterion:

$$\min(\lambda_1, \lambda_2) > threshold$$

This criterion improves the accuracy of corner detection and is particularly useful in applications requiring high-quality feature points, such as object tracking.

2.3 Mathematical Representation of Corner Detection

Corner detection algorithms typically involve analyzing the intensity changes in a small neighborhood around each pixel. The covariance matrix M of image gradients is calculated as:

$$M = \sum I_x^2 \sum I_x I_y \sum I_x I_y \sum I_y^2$$

where I_x and I_y are the image gradients in the x and y directions, respectively. The eigenvalues of M, λ_1 and λ_2, indicate the rate of change in intensity, with larger values signifying strong corner characteristics.

2.4 Algorithm Pseudocode

A general pseudocode for corner detection using the Harris method is as follows:

1. **Compute image gradients** I_x and I_y using a Sobel or similar operator.
2. **Calculate the covariance matrix** M for each pixel using I_x and I_y.
3. **Compute the Harris response** R based on the eigenvalues of M.
4. **Identify corners** by selecting pixels with high values of R above a set threshold.
5. **Apply non-maximum suppression** to retain only the strongest corners and remove redundant points.

2.5 Diagram of Corner Detection Process

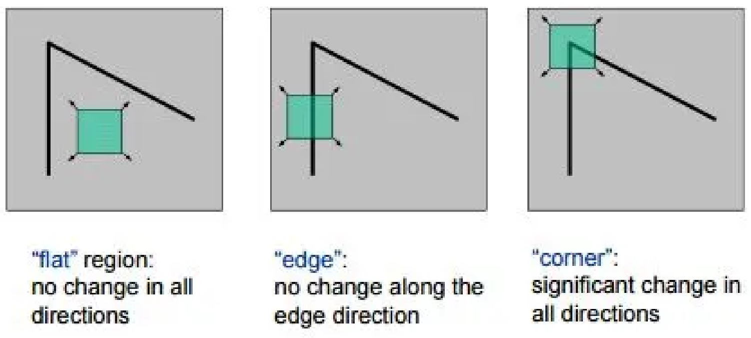
"flat" region:
no change in all directions
"edge":
no change along the edge direction
"corner":
significant change in all directions

Figure 2.1: Overview of the Corner Detection process, illustrating steps from gradient calculation to final corner selection.

2.6 Applications of Corner Detection

Corner detection is essential in several applications within image processing and computer vision:

- **Image Matching and Registration**: Corners serve as reliable keypoints for matching corresponding points between images, aiding in aligning images for tasks such as panoramic stitching or 3D reconstruction.
- **Object Tracking**: Corner points are used to track objects over a sequence of frames in videos, as they remain relatively stable and identifiable under varying viewpoints.
- **3D Reconstruction**: By matching corner points across multiple images, algorithms can reconstruct 3D structures and models of objects, providing depth and spatial information.
- **Feature-based Recognition**: Corners are often used as features in object recognition tasks where specific patterns or shapes need to be identified within an image.

Chapter 3

The Hough and Radon Transform

The Hough and Radon Transforms are mathematical techniques widely used in image processing and computer vision for detecting shapes, lines, and other structural elements within an image. Both transforms work by converting geometric structures in the spatial domain into parameters in a transformed domain, allowing for the detection of patterns and shapes that may be challenging to identify directly in an image. These transforms are foundational tools in applications such as medical imaging, object detection, and industrial inspection.

3.1 Fundamentals of the Hough Transform

The Hough Transform is a feature extraction technique used to detect simple shapes like lines, circles, and ellipses in images. It maps points in an image space to a parameter space, where shapes can be detected as peaks in the transform space. Originally developed for detecting straight lines, the Hough Transform has been extended to identify more complex shapes.

3.1.1 Line Detection with Hough Transform

To detect lines, the Hough Transform represents lines in terms of polar coordinates rather than the Cartesian form $y = mx + b$. In the Hough space, a line is defined by the parameters ρ (the perpendicular distance from the origin) and θ (the angle of the perpendicular line from the x-axis), as follows:

$$\rho = x \cos \theta + y \sin \theta$$

Each point in the image space (x, y) generates a sinusoidal curve in the Hough space, representing all possible lines passing through that point. When multiple points lie along the same line in the image, their curves intersect in the Hough space at a common (ρ, θ), forming a peak that can be detected to identify the line.

3.1.2 Circle Detection with Hough Transform

The Hough Transform can also be adapted for circle detection by parameterizing the equation of a circle. For a circle with radius r and center (a, b), the equation is:

$$(x - a)^2 + (y - b)^2 = r^2$$

This creates a three-dimensional Hough space with parameters a, b, and r. When an edge point in the image lies on a circle of radius r, it generates a circular set of possible centers (a, b) in the Hough space. Peaks in this space indicate the presence of circles with specific radii and centers.

3.2 Fundamentals of the Radon Transform

The Radon Transform is closely related to the Hough Transform and is primarily used to detect linear structures by integrating image intensities along specified directions. Unlike the Hough Transform, which is designed to detect discrete features, the Radon Transform is often used in continuous image analysis and is especially useful in tomographic reconstruction (e.g., CT scans).

For a two-dimensional image $f(x, y)$, the Radon Transform $R(\rho, \theta)$ is defined as:

$$R(\rho, \theta) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y)\, \delta(\rho - x\cos\theta - y\sin\theta)\, dx\, dy$$

where δ is the Dirac delta function. This integral computes the sum of image intensities along lines defined by ρ and θ, effectively converting a 2D image into a set of 1D projections. These projections represent the accumulation of pixel intensities along different directions.

3.3 Applications of the Hough and Radon Transform

Both the Hough and Radon Transforms have a wide range of applications across different fields. Their utility stems from their ability to detect shapes and patterns effectively, even in noisy or incomplete data:

- **Medical Imaging (Radon Transform)**: The Radon Transform is essential in computed tomography (CT) imaging, where it is used to reconstruct cross-sectional images of a patient's body from multiple projections, allowing for accurate diagnosis and analysis.
- **Lane Detection in Autonomous Vehicles (Hough Transform)**: The Hough Transform is commonly used for detecting lane boundaries in road images. By identifying straight lines that represent lanes, the algorithm assists in lane-keeping and navigation for autonomous vehicles.
- **Object Recognition in Industrial Inspection (Hough Transform)**: In automated manufacturing and inspection, the Hough Transform helps detect specific shapes (e.g., circles, lines) to verify the structural integrity of parts, such as identifying drilled holes or detecting defects.
- **Image Registration and Alignment (Radon Transform)**: The Radon Transform can assist in aligning and registering images by matching structures across different images, as it effectively captures the directional features that can be used for alignment.

3.4 Comparison of the Hough and Radon Transforms

While both the Hough and Radon Transforms detect geometric shapes, they have unique characteristics:

- **Hough Transform**: It is primarily a discrete feature detection method, effective for detecting defined shapes (e.g., lines, circles) in binary or edge-detected images. It is highly adaptable but computationally expensive, especially for shapes requiring multiple parameters.
- **Radon Transform**: This transform is used for continuous data and is particularly useful in applications requiring image reconstruction, like tomography. It provides projection data over continuous angles, making it suitable for tasks involving integration along lines or directions.

3.5 Diagram of the Hough and Radon Transform Processes

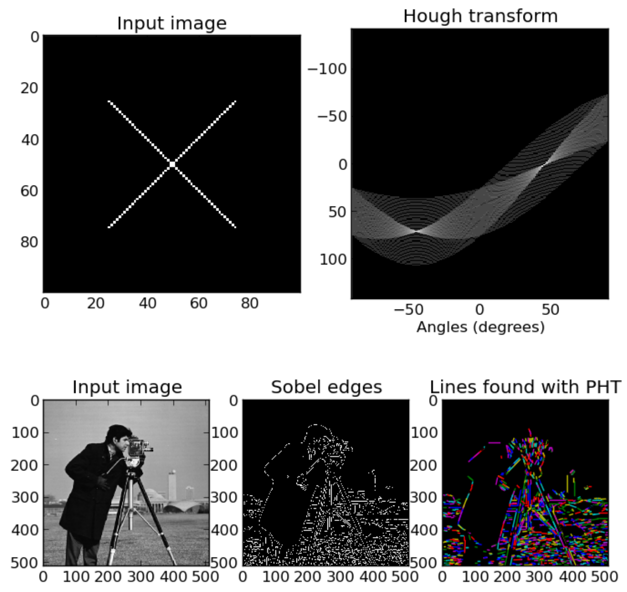

Figure 3.1: Illustration of the Hough Transform processes. The Hough Transform maps points in the spatial domain to parameter space for line and circle detection

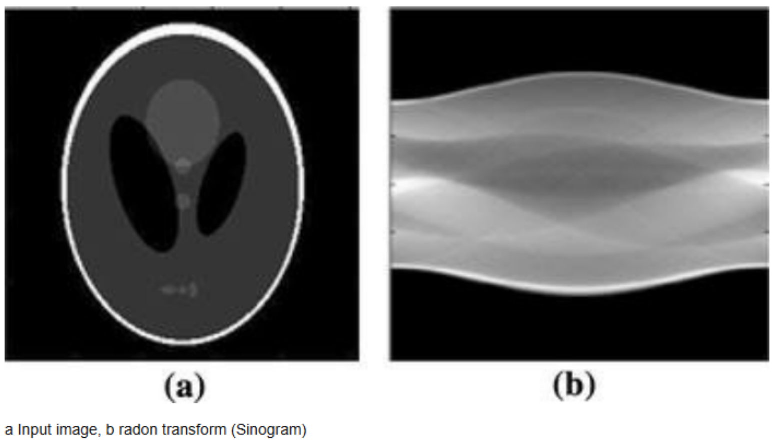

Figure 3.2: Illustration of Radon Transform processes. The Radon Transform accumulates intensity along lines for projection-based analysis.

3.6 Algorithm Pseudocode

A simplified pseudocode for line detection using the Hough Transform is as follows:

1. **Edge Detection**: Apply an edge detection algorithm (e.g., Canny) to identify edge points in the image.
2. **Parameter Space Accumulation**: For each edge point (x, y), calculate the corresponding (ρ, θ) values for a range of angles and increment the accumulator array.
3. **Detect Peaks**: Identify peaks in the accumulator array, which correspond to lines in the image space.
4. **Map Back to Image**: Convert the identified peaks in parameter space back to lines in the image.

The Radon Transform, being an integration-based approach, does not require an accumulator array. Instead, it computes projections for specific angles directly.

Chapter 4

Bayesian Classification

Bayesian Classification is a probabilistic approach to classification that applies Bayes' Theorem to make decisions based on conditional probabilities. In this approach, classification is performed by calculating the posterior probability for each class given a set of observed features, allowing for a decision based on the likelihood of each class. Bayesian classifiers are widely used in applications such as spam detection, medical diagnosis, and pattern recognition due to their simplicity, interpretability, and solid theoretical foundation.

4.1 Fundamentals of Bayesian Classification

The principle of Bayesian classification relies on Bayes' Theorem, which states that for events A and B,

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

where: - $P(A|B)$ is the posterior probability of A given B, - $P(B|A)$ is the likelihood of observing B given A, - $P(A)$ is the prior probability of A, and - $P(B)$ is the marginal probability of B.

In classification, A typically represents a particular class label, and B represents the observed data (features). The goal is to determine the class that maximizes the posterior probability $P(A|B)$, thereby making the most probable classification for the given features.

4.2 Types of Bayesian Classifiers

Several types of Bayesian classifiers exist, each tailored for different assumptions and applications:

4.2.1 Naive Bayes Classifier

The Naive Bayes classifier is one of the simplest Bayesian classifiers, which assumes that features are conditionally independent given the class label. This “naive” assumption allows for easy computation of the posterior probability as follows:

$$P(C|X) = \frac{P(C)\prod_{i=1}^{n} P(X_i|C)}{P(X)}$$

where C is the class label, $X = (X_1, X_2, \ldots, X_n)$ represents the feature vector, and $P(X_i|C)$ is the likelihood of each feature given the class. Despite its simplicity, Naive Bayes is surprisingly effective in many applications, particularly when the independence assumption is approximately valid.

4.2.2 Gaussian Naive Bayes

Gaussian Naive Bayes is a variant of Naive Bayes used for continuous data. Here, the likelihood $P(X_i|C)$ is assumed to follow a Gaussian (normal) distribution:

$$P(X_i|C) = \frac{1}{\sqrt{2\pi\sigma_C^2}} \exp\left(-\frac{(X_i - \mu_C)^2}{2\sigma_C^2}\right)$$

where μ_C and σ_C^2 are the mean and variance of the feature X_i for class C. This approach is particularly useful when dealing with real-valued features.

4.2.3 Bayesian Network Classifier

A Bayesian Network is a more complex Bayesian classifier that models dependencies between features, relaxing the independence assumption of Naive Bayes. Bayesian Networks represent the conditional dependencies between features as a directed acyclic graph (DAG), where nodes represent variables (features) and edges represent dependencies. This method can capture complex relationships between features, making it powerful for high-dimensional data where features are interdependent.

4.3 Mathematical Representation

Given an input feature vector $X = (X_1, X_2, \ldots, X_n)$ and a set of possible classes $C_1, C_2, \ldots, C_k$, Bayesian classification calculates the posterior probability for each class C_i as:

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}$$

The classifier assigns X to the class with the highest posterior probability:

$$\hat{C} = \arg\max_{C_i} P(C_i|X)$$

In practice, $P(X)$ is constant across all classes, so maximizing $P(C_i|X)$ is equivalent to maximizing the numerator $P(X|C_i)P(C_i)$.

4.4 Algorithm Pseudocode

The pseudocode for Bayesian classification using the Naive Bayes method is as follows:

1. **Compute prior probabilities** $P(C)$ for each class C based on the training data.
2. **Calculate likelihoods** $P(X_i|C)$ for each feature X_i given each class C using training data.
3. **For each new instance** X:
   (a) Calculate the posterior probability $P(C|X)$ for each class C as $P(C|X) \propto P(C) \prod_{i=1}^{n} P(X_i|C)$.
   (b) Assign X to the class with the highest posterior probability.

4.5 Diagram of Bayesian Classification Process

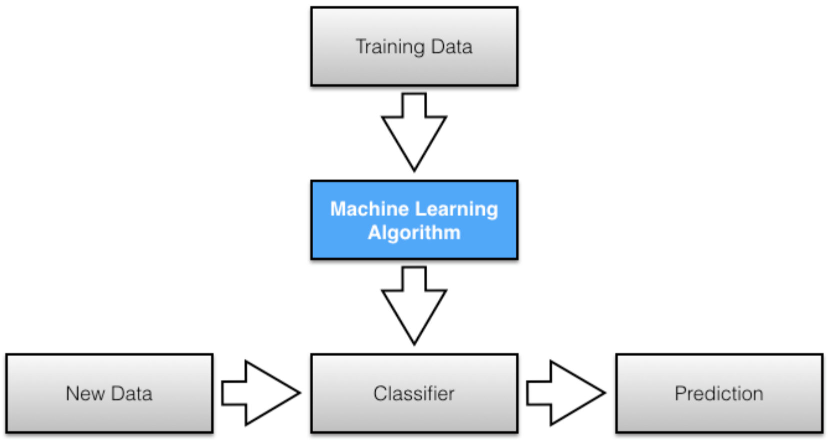

Figure 4.1: Illustration of the Bayesian Classification process, showing the steps from prior and likelihood calculation to final classification based on posterior probability.

4.6 Applications of Bayesian Classification

Bayesian classification is used across various fields due to its effectiveness in probabilistic reasoning:

– **Spam Filtering**: Bayesian classifiers, particularly Naive Bayes, are widely used in spam detection, classifying emails as spam or non-spam based on the probability of keywords and other features.

– **Medical Diagnosis**: Bayesian classifiers are employed in diagnosing diseases based on symptoms, medical history, and test results, enabling probabilistic assessments of disease likelihoods.

– **Text Classification**: In natural language processing, Bayesian classifiers help categorize text into topics or sentiments, relying on word frequencies and probabilities.

– **Image Recognition**: Bayesian methods, especially Bayesian Networks, are used in recognizing objects, faces, and scenes by modeling complex dependencies between image features.

Chapter 5

k-means clustering

5.1 Introduction to K-means Clustering

K-means clustering is an unsupervised learning algorithm widely used in data analysis and pattern recognition. In digital image processing, it is used to segment an image into distinct regions or clusters based on pixel intensity or color similarity. The goal is to partition the data (pixel values) into k clusters, where each pixel belongs to the cluster with the nearest mean.

5.2 K-means Algorithm Overview

- **Initialization:** Choose the number of clusters `k` and initialize `k` centroids randomly.
- **Assignment:** For each pixel in the image, calculate its distance from each centroid. Assign the pixel to the cluster with the closest centroid.
- **Update:** Recalculate the centroids by computing the mean of the pixels in each cluster.
- **Iteration:** Repeat the assignment and update steps until convergence (when centroids no longer change significantly) or until a specified number of iterations is reached.

5.3 K Means Clustering Algorithm

K Means is a clustering algorithm. Clustering algorithms are unsupervised algorithms which means that there is no labelled data available. It is used to identify different classes or clusters in the given data based on how similar the data is. Data points in the same group are more similar to other data points in that same group than those in other groups.
The working of the K-Means algorithm is explained in the below steps:

- **Step 1:** Select the number K to decide the number of clusters.
- **Step 2:** Select random K points or centroids. (It can be other from the input dataset).
- **Step 3:** Assign each data point to their closest centroid, which will form the predefined K clusters.
- **Step 4:** Calculate the variance and place a new centroid of each cluster.
- **Step 5:** Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.
- **Step 6:** If any reassignment occurs, then go to step-4 else go to FINISH.
- **Step 7:** The model is ready.

5.4 Mathematical Details

5.4.1 Distance Calculation

The distance between a pixel x_i and a centroid μ_j is often measured using the Euclidean distance formula:

$$d(x_i, \mu_j) = \sqrt{(x_{i1} - \mu_{j1})^2 + (x_{i2} - \mu_{j2})^2 + \cdots + (x_{im} - \mu_{jm})^2}$$

where m represents the pixel features (e.g., RGB color channels).

5.4.2 Cluster Assignment

Each pixel x_i is assigned to a cluster C_j based on the minimum distance:

$$C_j = \{x_i : \|x_i - \mu_j\| \leq \|x_i - \mu_l\| \, \forall l \in \{1, \ldots, k\}\}$$

5.4.3 Centroid Update

The centroid of each cluster j is updated to be the mean of all pixels assigned to that cluster:

$$\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_i} x_i$$

5.5 Mathematical Example of K Means Algorithm

Lets consider we have cluster points P1(1,3), P2(2,2), P3(5,8), P4(8,5), P5(3,9), P6(10,7), P7(3,3), P8(9,4), P9(3,7).

First, we take our K value as 3 and we assume that our Initial cluster centers are P7(3,3), P9(3,7), P8(9,4) as C1, C2, C3. We will find out the new centroids after 2 iterations for the above data points.

Step 1:
Find the distance between data points and Centroids. which data points have a minimum distance that points moved to the nearest cluster centroid.

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

Iteration 1:
Calculate the distance between data points and K (C1,C2,C3) C$_1P_1 \Rightarrow (3,3)(1,3) \Rightarrow$ $\sqrt{(1-3)^2 + (3-3)^2} \Rightarrow \sqrt{4} \Rightarrow 2$
$C_2P_1 \Rightarrow (3,7)(1,3) \Rightarrow \sqrt{(1-3)^2 + (3-7)^2} \Rightarrow \sqrt{20} \Rightarrow 4.5$
$C_3P_1 \Rightarrow (9,4)(1,3) \Rightarrow \sqrt{(1-9)^2 + (3-4)^2} \Rightarrow \sqrt{65} \Rightarrow 8.1$

For P_2,

C$_1P_2 \Rightarrow (3,3)(2,2) \Rightarrow \sqrt{(2-3)^2 + (2-3)^2} \Rightarrow \sqrt{2} \Rightarrow 1.4$
$C_2P_2 \Rightarrow (3,7)(2,2) \Rightarrow \sqrt{(2-3)^2 + (2-7)^2} \Rightarrow \sqrt{26} \Rightarrow 5.1$
$C_3P_2 \Rightarrow (9,4)(2,2) \Rightarrow \sqrt{(2-9)^2 + (2-4)^2} \Rightarrow \sqrt{53} \Rightarrow 7.3$

For P_3,

C$_1P_2 \Rightarrow (3,3)(5,8) \Rightarrow \sqrt{(5-3)^2 + (8-3)^2} \Rightarrow \sqrt{29} \Rightarrow 5.3$
$C_2P_2 \Rightarrow (3,7)(5,8) \Rightarrow \sqrt{(5-3)^2 + (8-7)^2} \Rightarrow \sqrt{5} \Rightarrow 2.2$
$C_3P_2 \Rightarrow (9,4)(5,8) \Rightarrow \sqrt{(5-9)^2 + (8-4)^2} \Rightarrow \sqrt{32} \Rightarrow 5.7$

Similarly for other distances..

Data Points	Centroid (3,3)	Centroid (3,7)	Centroid (9,4)	Cluster
P1(1,3)	2	4.5	8.1	C1
P2(2,2)	1.4	5.1	7.3	C1
P3(5,8)	5.3	2.2	5.7	C2
P4(8,5)	5.4	5.4	5.1	C3
P5(3,9)	6	2	7.9	C2
P6(10,7)	8.1	7	3.2	C3
P7(3,3)	0	4	6.1	C1
P8(9,4)	6.1	6.7	0	C3
P9(3,7)	4	0	6.7	C2

Table 5.1: Distances from Centroids

Cluster 1 $\Rightarrow P_1(1,3), P_2(2,2), P_7(3,3)$
$Cluster2 \Rightarrow P_3(5,8), P_5(3,9), P_9(3,7)$
$Cluster3 \Rightarrow P_4(8,5), P_6(10,7), P_8(9,4)$

Now, We re-compute the new clusters and the new cluster center is computed by taking the mean of all the points contained in that particular cluster.

New center of Cluster 1 $\Rightarrow \frac{(1+2+3)}{3}, \frac{(3+2+3)}{3} \Rightarrow 2, 2.7$
$NewcenterofCluster2 \Rightarrow \frac{(5+3+3)}{3}, \frac{(8+9+7)}{3} \Rightarrow 3.7, 8$
$NewcenterofCluster3 \Rightarrow \frac{(8+10+9)}{3}, \frac{(5+7+4)}{3} \Rightarrow 9, 5.3$

Iteration 1 is over. Now, let us take our new center points and repeat the same steps which are to calculate the distance between data points and new center points with the Euclidean formula and find cluster groups.

Iteration 2:

Calclate the distance between data points and K $C_1, C_2, C_3)$

$C_1(2,2.7), C_2(3.7,8), C_3(9,5.3)$

C$_1P_1 \Rightarrow (2,2.7)(1,3) \Rightarrow \sqrt{(1-2)^2+(3-2.7)^2} \Rightarrow \sqrt{1.1} \Rightarrow 1.0$
$C_2P_1 \Rightarrow (3.7,8)(1,3) \Rightarrow \sqrt{(1-3.7)^2+(3-8)^2} \Rightarrow \sqrt{32.29} \Rightarrow 4.5$
$C_3P_1 \Rightarrow (9,5.3)(1,3) \Rightarrow \sqrt{(1-9)^2+(3-5.3)^2} \Rightarrow \sqrt{69.29} \Rightarrow 8.3$

Similarly for other distances..

Cluster 1 $\Rightarrow P_1(1,3), P_2(2,2), P_7(3,3)$
$Cluster2 \Rightarrow P_3(5,8), P_5(3,9), P_9(3,7)$

Data Points	Centroid (2,2.7)	Centroid (3.7,8)	Centroid (9,5.3)	Cluster
P1(1,3)	1.0	4.5	8.3	C1
P2(2,2)	0.7	6.2	7.7	C1
P3(5,8)	6.1	1.3	4.8	C2
P4(8,5)	6.4	5.2	1.0	C3
P5(3,9)	6.4	1.2	7.0	C2
P6(10,7)	9.1	6.4	1.9	C3
P7(3,3)	1.0	5.0	6.4	C1
P8(9,4)	7.1	6.6	1.3	C3
P9(3,7)	4.4	1.2	6.2	C2

Table 5.2: Distances from Centroids

$Cluster3 \Rightarrow P_4(8,5), P_6(10,7), P_8(9,4)$

Center of Cluster 1 $\Rightarrow \frac{(1+2+3)}{3}, \frac{(3+2+3)}{3} \Rightarrow 2, 2.7$
$CenterofCluster2 \Rightarrow \frac{(5+3+3)}{3}, \frac{(8+9+7)}{3} \Rightarrow 3.7, 8$
$CenterofCluster3 \Rightarrow \frac{(8+10+9)}{3}, \frac{(5+7+4)}{3} \Rightarrow 9, 5.3$

We got the same centroid and cluster groups which indicates that this dataset has only 2 groups. K-Means clustering stops iteration because of the same cluster repeating so no need to continue iteration and display the last iteration as the best cluster groups for this dataset.

Chapter 6

Support Vector Machine

Support Vector Machine (SVM) is a powerful supervised learning algorithm commonly used for classification and regression tasks in machine learning. SVM works by finding an optimal hyperplane that best separates data points of different classes in a high-dimensional space. This approach is highly effective for binary classification problems, where the goal is to separate two classes with the maximum possible margin, making SVM an ideal choice for applications in image classification, text categorization, and bioinformatics.

6.1 Fundamentals of Support Vector Machine

The fundamental concept of SVM is to transform the input data into a higher-dimensional space where a linear separator (hyperplane) can be used to distinguish between classes. This transformation is often achieved using kernel functions, enabling SVM to handle non-linear classification tasks effectively. The optimal hyperplane is defined as the one that maximizes the margin between the two classes, thereby minimizing classification error.

6.2 Types of Support Vector Machine

There are several variations of SVM, each tailored to different types of data and problem requirements:

6.2.1 Linear SVM

Linear SVM is used for linearly separable data, where a single straight line (or hyperplane in higher dimensions) can effectively separate the classes. The objective is to find the hyperplane that maximizes the margin between the two classes.

6.2.2 Non-Linear SVM

Non-linear SVM is used for data that is not linearly separable. In this case, SVM utilizes kernel functions to transform the data into a higher-dimensional space where a linear separator can be applied. Common kernels include the polynomial, radial basis function (RBF), and sigmoid kernels.

6.3 Soft Margin and Hard Margin

- **Hard Margin**: Requires a perfectly linear separation between classes, which is often impractical with noisy data.
- **Soft Margin**: Introduces slack variables to allow some misclassification, making SVM adaptable to real-world datasets by balancing margin maximization with error minimization.

6.4 Kernel Trick and Hyperparameter Tuning

The kernel trick allows SVM to handle non-linearly separable data. Common kernel functions include:

- **Linear Kernel**: Suitable for linearly separable data.
- **Polynomial Kernel**: Maps data to a higher dimension based on a polynomial function.
- **Radial Basis Function (RBF) Kernel**: Maps data to an infinite-dimensional space, effectively separating complex datasets.
- **Sigmoid Kernel**: Often used in neural networks, enabling SVM to mimic neural network functionality.

Hyperparameters, such as C and *gamma*, are key for tuning the SVM model:

- **C Parameter**: Controls the trade-off between maximizing the margin and minimizing classification errors. A higher C value emphasizes correct classification, while a lower C value prioritizes a larger margin.

– **Gamma Parameter** (in RBF kernel): Determines the influence of a single training example. Higher gamma values create tighter decision boundaries around individual points, while lower values create broader decision regions.

6.5 Mathematical Representation of SVM

The goal of SVM is to find the hyperplane that maximizes the margin between classes. For a given set of training data (x_i, y_i), where $y_i \in \{-1, +1\}$, the optimization problem can be defined as:

$$\min \frac{1}{2}\|w\|^2$$

subject to

$$y_i(w \cdot x_i + b) \geq 1, \quad \forall i$$

where w is the weight vector, b is the bias term, and x_i represents the input features. The solution to this optimization problem gives the parameters of the hyperplane that maximizes the margin.

6.6 Algorithm Pseudocode

A general pseudocode for the Support Vector Machine training process is as follows:

1. **Initialize parameters** for the weight vector w and bias b.
2. **Transform data** (if necessary) using a kernel function to map the input features to a higher-dimensional space.
3. **Optimize the hyperplane** by solving the SVM objective function to find w and b.
4. **Classify new data** by computing the decision function $f(x) = w \cdot x + b$. The sign of $f(x)$ determines the class label.

6.7 Diagram of SVM Process

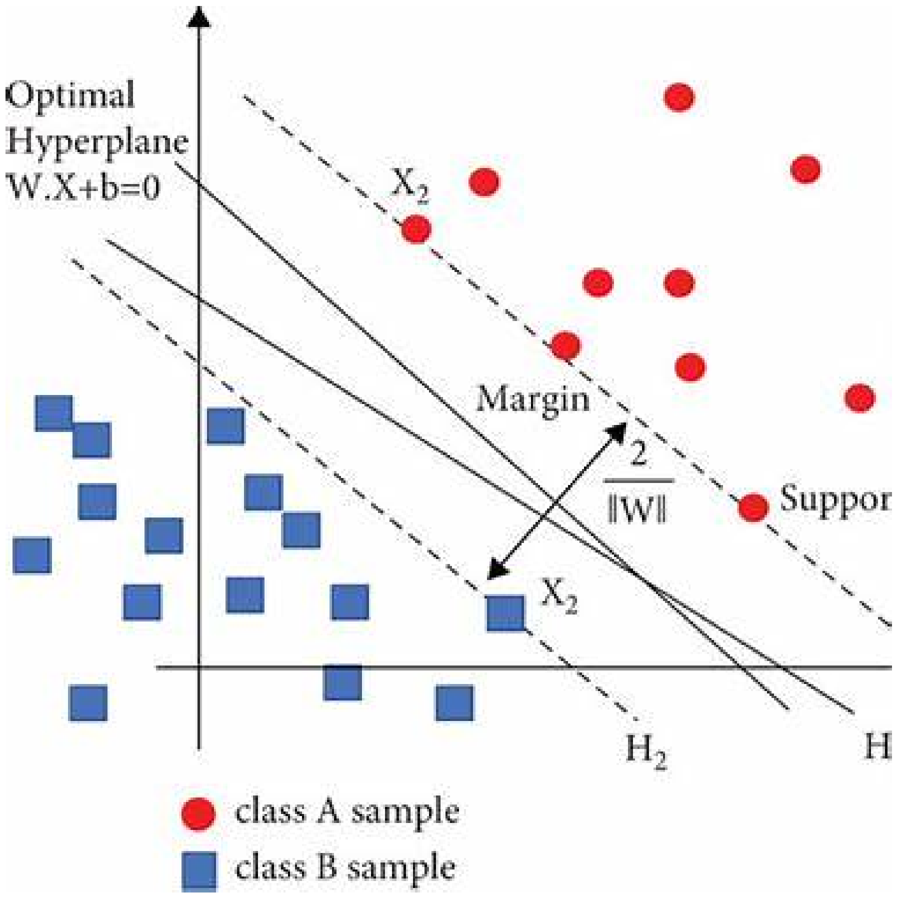

Figure 6.1: Overview of the Support Vector Machine process, illustrating the separation of data points using an optimal hyperplane$_1$.

6.8 Applications of Support Vector Machine

Support Vector Machine is used in various domains due to its effectiveness in handling high-dimensional and complex data:

- **Image Classification**: SVM is widely used for image classification tasks, such as recognizing objects, faces, and handwriting.
- **Text Categorization**: SVM is popular for categorizing text into predefined topics, such as spam detection, sentiment analysis, and document classification.
- **Bioinformatics**: SVM is applied in bioinformatics for tasks like gene classification, protein structure prediction, and cancer diagnosis.
- **Financial Analysis**: SVM is used to predict stock price trends and for credit

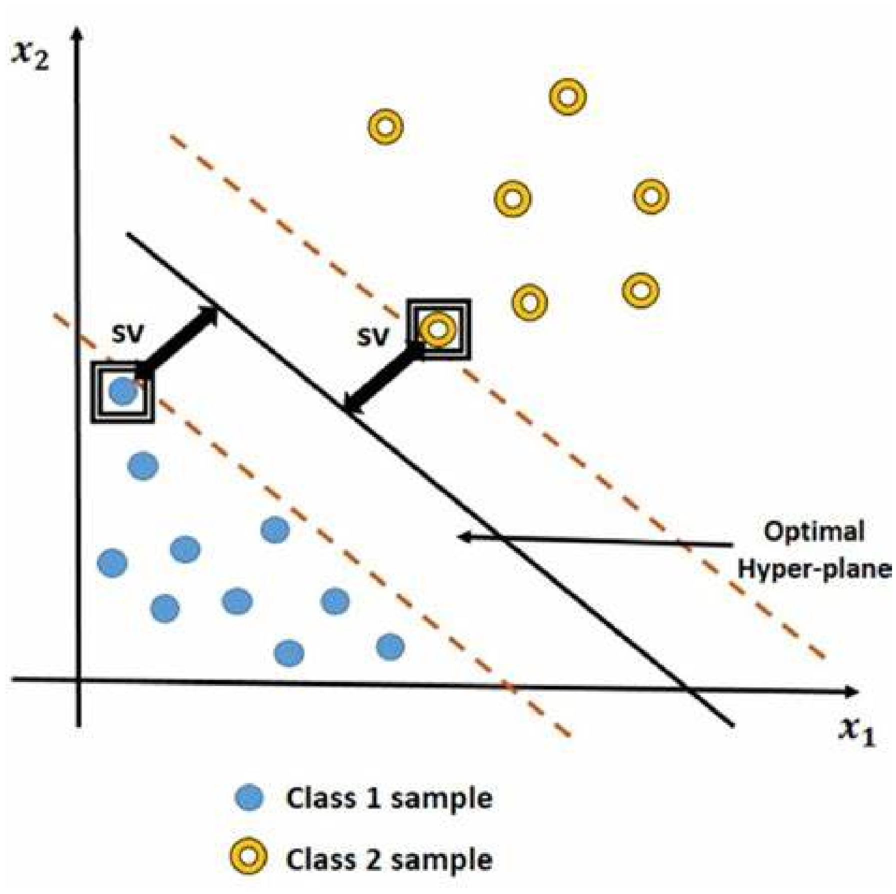

Figure 6.2: Overview of the Support Vector Machine process, illustrating the separation of data points using an optimal hyperplane$_2$.

scoring by classifying financial data.

– **Speech Recognition**: SVM assists in speech recognition by classifying audio signals and identifying different phonetic units.

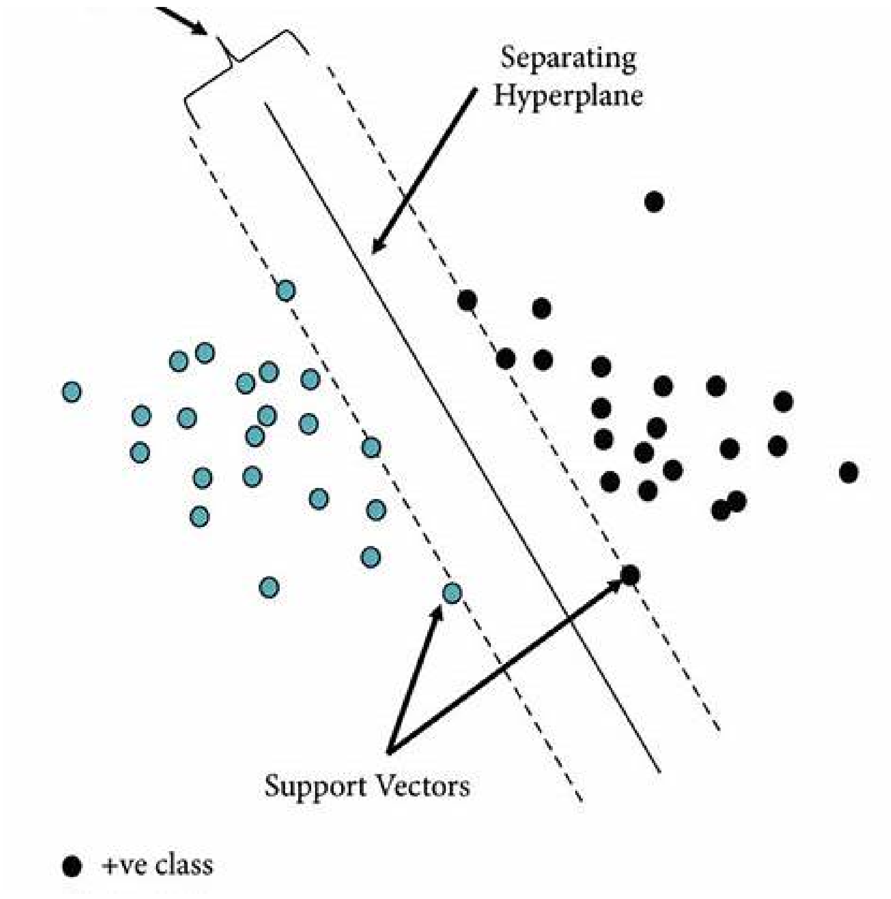

Figure 6.3: Overview of the Support Vector Machine process, illustrating the separation of data points using an optimal hyperplane$_3$.

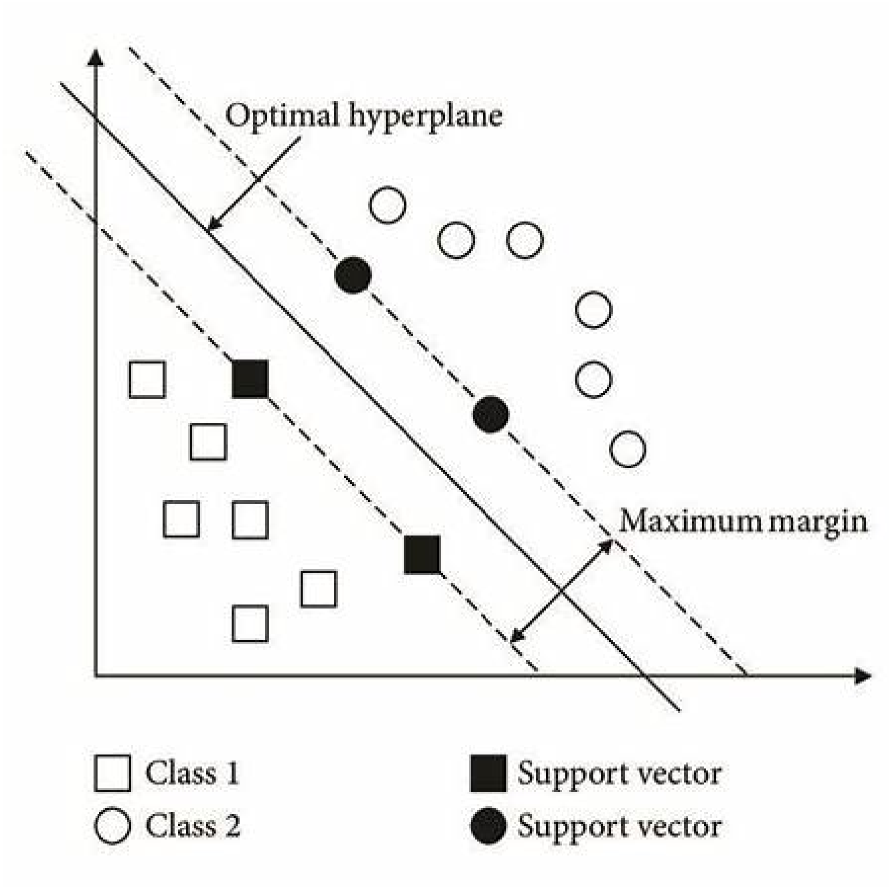

Figure 6.4: Overview of the Support Vector Machine process, illustrating the separation of data points using an optimal hyperplane$_4$.