HiLCoE

Artificial Intelligence (CS488)

GROUP ASSIGNMENT 1

Title: Solve Machine Learning Problems

GROUP MEMBERS

1. **LEUL YARED** **ID-YN8503**
2. **MERYEM ABDELLA** **ID-BN7468**
3. **MICHAEL DEMISSEW** **ID-CB2419**
4. **MICHAEL GETU** **ID-ML2612**
5. **MICHAEL WAGAYE** **ID-YG6812**

SECTION: DRB2102-B
SUBMITTED TO: Mr. Fantahun
SUBMITTING DATE: 17/08/2024

Acknowledgment

We would like to thank our instructor, Mr. Fantahun, for his incredible and interactive teaching methods. We would also like to thank him for giving us this assignment, where we learned several critical concepts related to machine learning, including:

- Understanding the impact of tree depth on overfitting and underfitting in decision trees.
- Exploring how the choice of k and distance metric affects the performance of the K-Nearest Neighbors (KNN) classifier.
- Gaining insight into the workings and limitations of the perceptron learning algorithm.
- Evaluating the strengths and weaknesses of various classifiers and how they perform on the same dataset.
- Understanding the mechanics of k-means clustering and how to assess clustering performance.
- Comparing the differences between k-means and k-medoids clustering, and evaluating their performance on the same dataset.

This assignment has made us greatly realize all these critical topics, through an opportunity to use these concepts in practice.

- Group members

Introduction

An algorithm refers to a well-defined procedure or set of rules to solve a particular problem.

- **Learning Algorithm:** is an algorithm which, over time, performs better after experiencing or training with data. It is used to establish patterns from data and make predictions or decisions on new inputs.
  It can be broadly categorized into supervised, unsupervised, semi-supervised, and reinforcement learning based on the nature of the input data and learning process.

  - *Supervised Learning:* model gets trained on a labeled dataset wherein each example provided for training is associated with an output label. The model learns a mapping from inputs to the desired outputs.
  - *Unsupervised learning:* model gets trained on data without explicit labels. The aim is to find underlying structure in a dataset; often this is done through clustering or association.

Overfitting: An error occurring during modeling when a learning algorithm fits noise in a training dataset rather than underlying patterns in data; generally, it generalizes poorly to new data.
Underfitting: A situation wherein a learning algorithm fails to capture the underlying pattern in a data set, usually due to a too-simple model. It results in poor performance on both the training and test data.

Decision Tree

A decision tree is one of the supervised learning algorithms used in both classification and regression tasks. It works by modeling decisions and their possible results in a tree-like structure, where each internal node represents an attribute test, every branch is the result of the test performed, and every leaf node class label or output.

K-Nearest Neighbors (KNN)

K-Nearest Neighbors is among the most simplistic, non-parametric, supervised learning algorithms applied to classification and regression. Here, the algorithm assigns the majority class for the data point among its k nearest neighbors in the feature space, where k is user-defined.

- ***Euclidean Distance*** is the straight-line distance between two points in a Euclidean space, calculated using the Pythagorean theorem. It's the most direct path between two points.

- ***Manhattan Distance (or Taxicab Distance)*** is the distance between two points measured along axes at right angles. It's like how a taxi would drive in a city grid, only moving horizontally or vertically.

Perceptron Learning Algorithm

Perceptron is the simplest neural network, linear binary classifier, and hence forms the basis for all other neural networks. It is an example of a supervised learning algorithm, in that it learns by updating its weights iteratively based on misclassification of training samples

Clustering

Clustering is another unsupervised learning technique used to partition a set of points into clusters so that points in one cluster are more similar to each other than to points in another cluster.

- *K-Means Clustering:* perhaps the most popular partition-based clustering algorithm, which has the aim of dividing a dataset into k clusters, where every data point belongs to the cluster having the nearest mean (centroid).
- *K-Medoids Clustering:* is a variation of k-means clustering that is more resistant to noise and outliers. Instead of taking the mean to represent each cluster, k-medoids use actual data points, called medoids, as the center of clusters.

Performance Metrics

Performance metrics are measurements used to evaluate the effectiveness of a machine learning model, particularly in classification tasks.

Accuracy: ratio of correctly predicted instances to total number of instances.

Accuracy = (True Positives + True Negatives) / Total Instances

It gives an overall measure of how often the model is correct, but it can be misleading if the data is imbalanced.

Precision: ratio of correctly predicted positive instances to total predicted positive instances.

Precision = (True Positives / (True Positives + False Positives)

It answers the question *"Of all the positive predictions, how many were actually correct?"*

Recall (Sensitivity or True Positive Rate): ratio of correctly predicted positive instances to all instances that were actually positive.

Recall = True Positives / (True Positives + False Negatives)

It answers the question *"Of all the actual positives, how many were correctly predicted?"*

F1-Score: The harmonic mean of Precision and Recall, providing a balance between the two.

F1-Score = 2 × ((Precision × Recall) / (Precision + Recall))

It's useful when you need a balance between Precision and Recall, especially with imbalanced datasets.

Confusion Matrix: A table that shows how well a classification model performs by comparing predicted vs. actual values, summarizing true positives, true negatives, false positives, and false negatives.

Silhouette Score: A measure for evaluating clustering quality, indicating how well-separated and cohesive the clusters are. Scores range from -1 (bad clustering) to 1 (good clustering).

This report provides an in-depth exploration of the topics discussed above, accompanied by a practical implementation using a real dataset.

Table of Content

Problems

***Problem #1*:** Decision Tree Depth Optimization

Task: Train a decision tree classifier on the Iris dataset and optimize its performance by adjusting the maximum depth of the tree.

Instructions:

1. Split the dataset into training and testing sets.
2. Train a decision tree with various maximum depths.
3. Evaluate the performance of each model using accuracy, precision, recall, and F1-score.
4. Visualize the decision boundaries of the best-performing model.
5. Plot the performance metrics against the tree depth to analyze the impact.

Problem #2: K-Nearest Neighbors Hyperparameter Tuning

Task: Implement a k-nearest neighbors (KNN) classifier on the Iris dataset and optimize the number of neighbors (k) and distance metrics (e.g. Euclidean, Manhattan).

Instructions:

1. Split the dataset into training and testing sets.
2. Train multiple KNN classifiers with different values of k and distance metrics.
3. Evaluate the models using accuracy, precision, recall, and F1-score.
4. Visualize the decision boundaries for different k values.
5. Plot performance metrics against different k values and distance metrics.

Problem #3: Perceptron Learning Algorithm

Task: Train a perceptron classifier on the Iris dataset and analyze its performance.

Instructions:

1. Split the dataset into training and testing sets.
2. Train a perceptron classifier on the training set.
3. Evaluate the model using accuracy, precision, recall, and F1-score.
4. Visualize the decision boundaries of the perceptron.
5. Plot the convergence of the perceptron learning algorithm over iterations.

Problem #4: Comparing Decision Tree, KNN, and Perceptron

Task: Compare the performance of decision tree, KNN, and perceptron classifiers on the Iris dataset.

Instructions:

1. Split the dataset into training and testing sets.
2. Train a decision tree, KNN, & perceptron classifier on training set.
3. Evaluate each model using accuracy, precision, recall, and F1-score.
4. Visualize the decision boundaries for all three classifiers.
5. Plot the performance metrics for each classifier.

Problem #5: K-Means Clustering and Visualization

Task: Apply k-means clustering to the Iris dataset and visualize the results.

Instructions:

1. Normalize the Iris dataset.
2. Apply k-means clustering with k=3.
3. Visualize the cluster assignments and the centroids in 2D and 3D plots.
4. Compare the cluster assignments with the actual class labels.
5. Plot the silhouette scores to evaluate the quality of the clusters.

Problem #6: K-Medoids Clustering and Comparison with K-Means

Task: Apply k-medoids clustering to the Iris dataset and compare it with k-means clustering.

Instructions:

1. Normalize the Iris dataset.
2. Apply k-medoids clustering with k=3.
3. Visualize cluster assignments and the medoids in 2D and 3D plots.
4. Compare the cluster assignments with the actual class labels.
5. Compare the performance of k-means and k-medoids using silhouette scores and other clustering metrics.

Problem #1: Decision Tree Depth Optimization

Source Code: **https://tinyurl.com/AI-Problems1**

Source Code Explanation:

- ***Dataset Splitting:***

  The code correctly splits the dataset into training (70%) and testing (30%) sets using *train_test_split*, ensuring that the model's performance is evaluated on unseen data.

- ***Training with Various Depths:***

  A loop iterates over various maximum depths (from 1 to 15) to train the decision tree classifiers, which is aligned with the task of experimenting with different depths.

- ***Performance Evaluation:***

  The code calculates the required performance metrics: accuracy, precision, recall, and F1-score for each depth. It uses the average='*macro*' argument, which is appropriate for multi-class classification problems like the Iris dataset.

  It also prints these metrics for each depth, making it easy to observe how performance changes with depth.

- ***Visualization of Decision Boundaries:***

  The code identifies the best-performing model based on the highest F1-score and visualizes its decision boundaries in two ways:

  Using PCA to reduce the features to 2D for visualization.

  Using only the first two features of the dataset to visualize the decision boundaries.

  I.e Unlike PCA this ensures clarity visualizing decision boundaries since it doesn't obscure some details due to dimensionality reduction.

- ***Performance Metrics Plotting:***

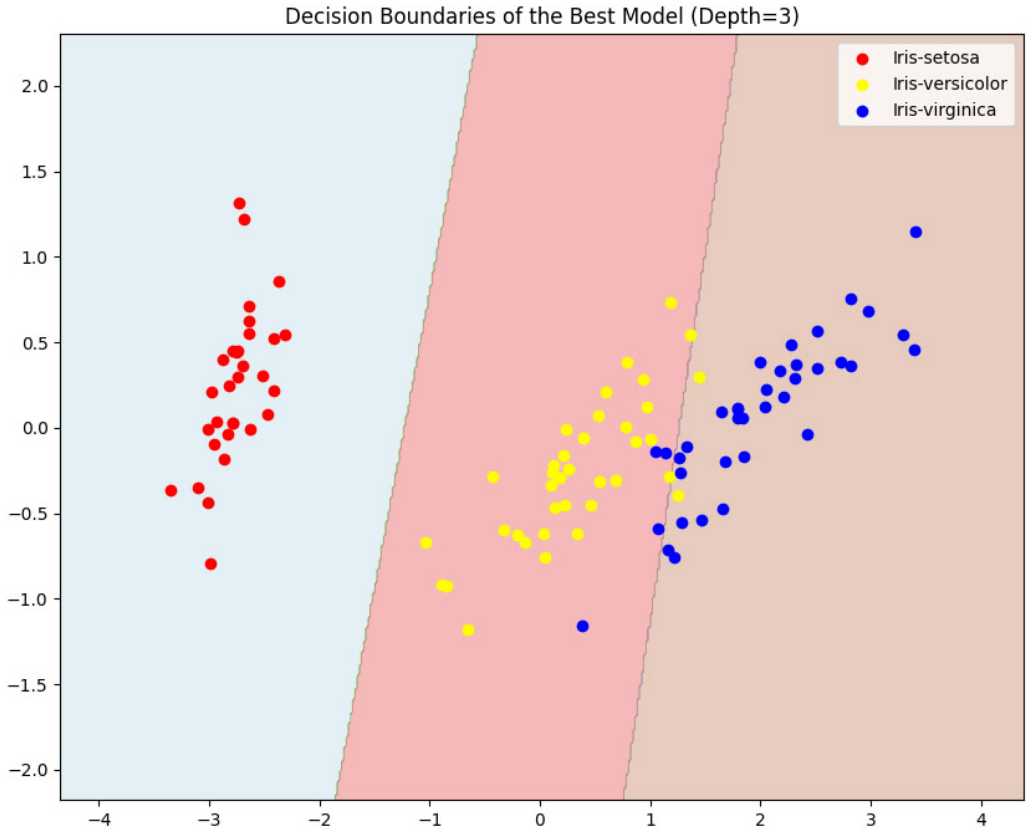
  The code plots the performance metrics (accuracy, precision, recall, and F1-score) against tree depth, which is crucial for analyzing the impact of tree depth on model performance, particularly in terms of overfitting and underfitting.

  - ***Shallow Trees (Low Depth):*** Lead to underfitting due to the model being too simple.
  - ***Deep Trees (High Depth):*** Lead to overfitting due to the model being overly complex.
  - ***Optimal Tree Depth:*** Balances complexity and generalization, minimizing both bias (underfitting) and variance (overfitting).

Outputs:

1. Visualization of decision boundary by the use of PCA
2. Visualization of decision boundary by the use of explictly the first two features
3. Performance metrics Vs Tree Depth

1. Visualization of decision boundary by the use of PCA

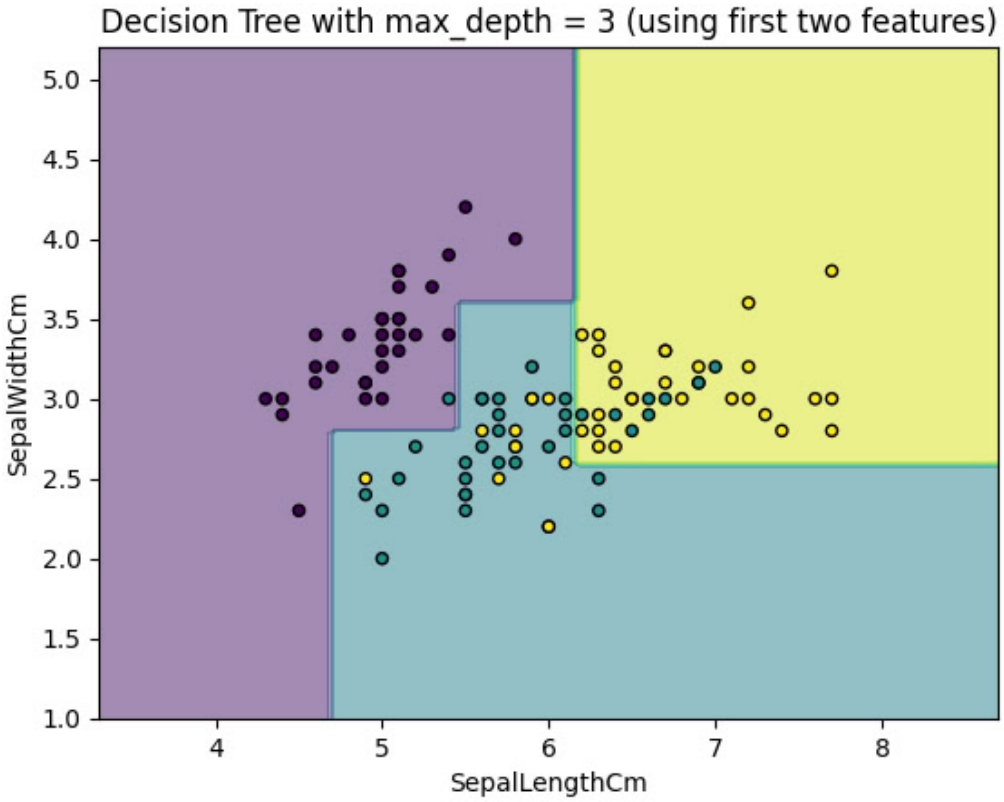


This graph depicts the decision boundaries of a decision tree classifier with a maximum depth of 3, trained using the Iris dataset. Here's what it shows:

- ***Axes:*** represent features of dataset reduced using PCA for visualization.
- ***Colored Regions:*** The different colored regions show the decision boundaries where the classifier predicts each Iris species:
  - ***Light blue:*** Iris-setosa
  - ***Pink***: Iris-versicolor
  - ***Light purple***: Iris-virginica
- ***Data Points:*** dots are actual data points from the Iris dataset.
- ***Decision Boundaries:*** The lines separating the regions are the decision boundaries created by the decision tree. These boundaries are where the decision tree splits the data to classify the points into different classes. Because the tree is of depth 3, the boundaries are relatively simple, consisting of horizontal and vertical lines.
- ***Insight Gained:***
  - ***Model Simplicity:*** The decision boundaries are relatively simple and linear, reflecting a low-complexity model.
  - ***Class Separation:*** The model separates the classes well, indicating that depth 3 effectively captures the data's structure without overfitting.

PCA (Principal Component Analysis) reduces the number of features in a dataset while preserving the most important information. It transforms the original features into a smaller set of new, uncorrelated features called ***principal components.*** This simplifies data, and reduces noise.

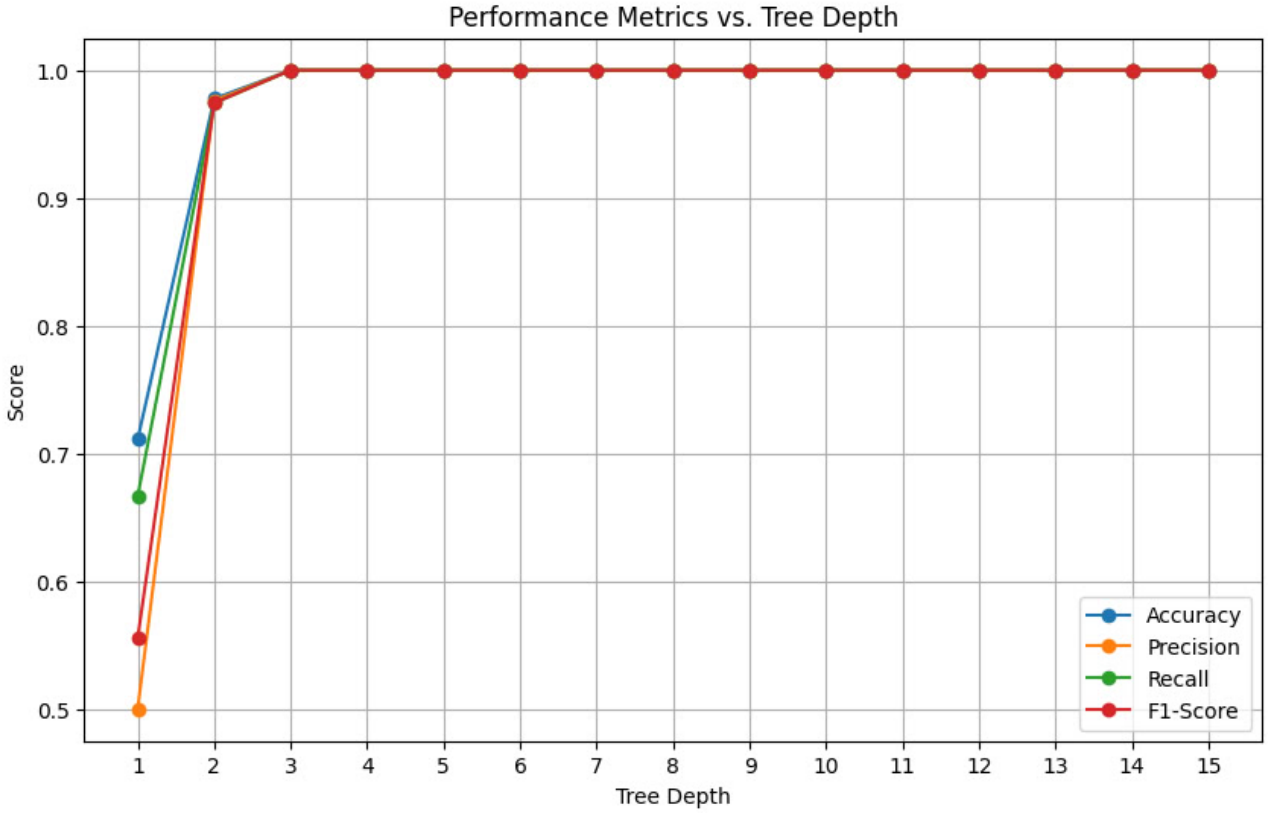
2. Visualization of decision boundary by the use of explictly the first two features



The graph shows the decision boundaries of a decision tree classifier trained on the Iris dataset using only the first two features: *Sepal Length and Sepal Width*, with a maximum depth of 3. Here's what it depicts:

- ***Axes:*** ***X-axis:*** Sepal Length in centimeters
  Y-axis: Sepal Width in centimeters
- ***Decision Boundaries:*** The colored regions indicate where the decision tree predicts each Iris species:
  Purple: Iris-setosa
  Yellow: Iris-versicolor
  Blue: Iris-virginica
- ***Data Points:*** The scatter points represent the actual data, colored according to their true class labels. The colors match the decision regions to show correct classifications.
- ***Model Behavior:*** The plot illustrates how the decision tree uses the two features to split the space into regions, aiming to classify the Iris species based on Sepal Length and Width. The blocky nature of the boundaries reflects the decision tree's tendency to create axis-aligned splits.
- ***Insight Gained:***
  Feature Influence: By using only the first two features, we see how these specific features contribute to class separation.
  Boundary Sharpness: The boundaries are clear and straightforward, demonstrating the decision tree's decision-making process with limited feature input.

3. Performance metrics Vs Tree Depth



The plots show how different performance metrics (accuracy, precision, recall, F1-score) vary with tree depth showing potential overfitting or underfitting.

- ***Axes:*** ***X-axis:*** Tree Depth
  Y-axis: Scores

- ***Insight Gained:***

  Underfitting: At very shallow depths (1-2), the model is underfitting. The decision tree is too simple to capture the complexities of the Iris dataset, resulting in lower scores across all metrics.

  Optimal Depth: The performance metrics rapidly improve as the tree depth increases to 3. At these depths, the decision tree is complex enough to capture the patterns in the data, leading to perfect or near-perfect classification performance.

  This suggests that a depth of 3 is optimal for this dataset, achieving a balance between model complexity and performance.

  No Overfitting: Unlike more complex datasets, this graph shows no sign of overfitting as the tree depth increases. All metrics remain at their maximum values beyond a depth of 4, indicating that the model's performance is stable and does not degrade with increased complexity.

Problem #2: K-Nearest Neighbors Hyperparameter Tuning

Source Code: **https://tinyurl.com/AI-Problems2**

Source Code Explanation:

- ***Split the dataset into training and testing sets***

  The code uses *train_test_split* to divide the dataset into training and testing sets, ensuring a 70-30 split.

- ***Train multiple KNN classifiers with different values of k and distance metrics***

  The code tests a variety of values for k (1, 3, 5, 7, 9, 11) and distance metrics (*euclidean and manhattan*). The approach of using a loop to train and evaluate the model across different combinations is efficient and meets the requirement.

- ***Evaluate the models using accuracy, precision, recall, and F1-score***

  The code correctly computes accuracy, precision, recall, and F1-score for each model configuration. The results are stored in a dictionary and converted into DataFrame, which is easier to analyze and plot results.

- ***Visualize the decision boundaries for different k values***

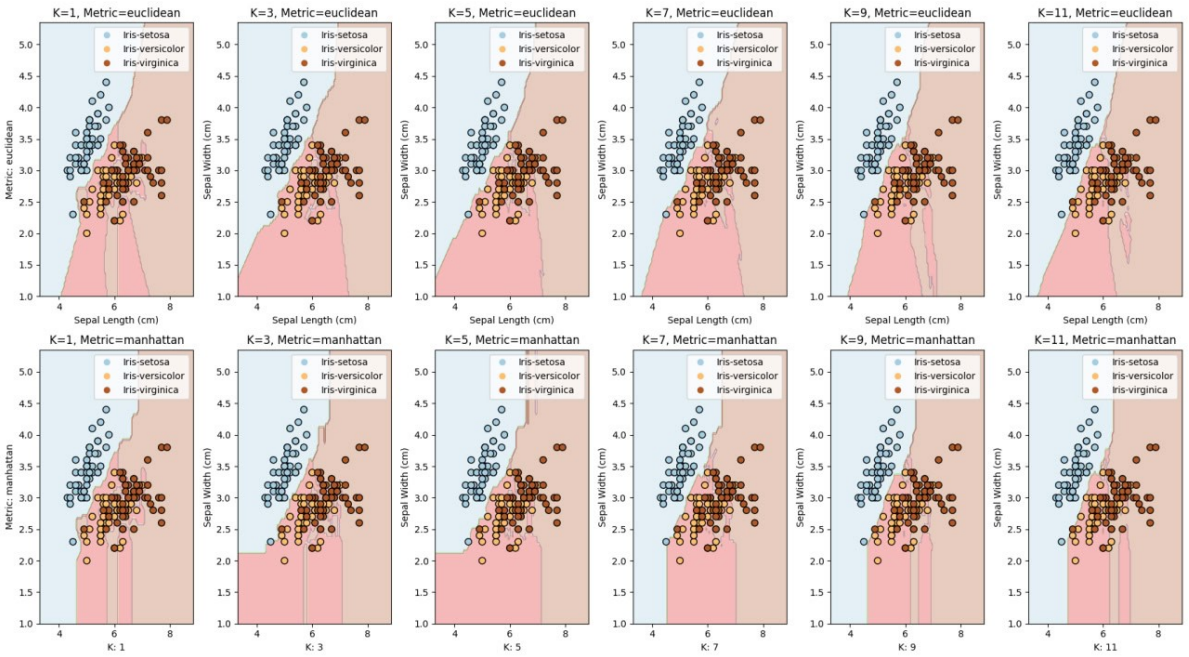
  The code includes a function *plot_decision_boundaries()* to visualize decision boundaries for different values of k and distance metrics. The plots are generated for the first two features (*Sepal Length and Sepal Width*), which is a reasonable approach for visualizing in 2D. However, this visualization only uses two features, which might not fully capture the model's behavior if the other two features are significant.

- ***Plot performance metrics against different k values and distance metrics***

  The code generates separate plots for each performance metric (accuracy, precision, recall, F1 score) against the values of k and different distance metrics. The plots are well-labeled, and the use of custom colors for each distance metric enhances readability.

Output:

1. Visualization of decision boundaries of a K-Nearest Neighbors (KNN) classifier



The graph depicts the decision boundaries of a K-Nearest Neighbors (KNN) classifier on the Iris dataset, using different values of k (number of neighbors) and two different distance metrics: Euclidean and Manhattan. Each subplot corresponds to a specific combination of k and distance metric.

The top row represents decision boundaries using the Euclidean distance metric.

The bottom row represents decision boundaries using the Manhattan distance metric.

Each column corresponds to a specific value of k (1, 3, 5, 7, 9, 11).

- ***Background Colors:*** The different colored regions in the background represent the decision boundaries of the KNN classifier. Each color indicates the predicted class label for that region.

  There are three colors, corresponding to the three species of Iris flowers: *Iris-setosa* (light blue), *Iris-versicolor* (pink), and *Iris-virginica* (light purple).
- ***Data Points:*** The scatter plots display the actual data points from the Iris dataset, with each point colored according to its true class label.

  The circles are data points, and the three classes are represented by different colors.
- **Insight Gained:**

  Effect of k (Number of Neighbors):

  As k increases, the decision boundaries tend to become smoother. With lower k values, the model is more sensitive to individual data points, resulting in more jagged and complex boundaries.

  For instance, at k=1, the model perfectly fits the training data, resulting in highly irregular boundaries. As k increases to 11, the boundaries become smoother and more generalized.

  Effect of Distance Metric:

  The top row (Euclidean) shows smoother, more rounded boundaries compared to the bottom row (Manhattan), which tends to produce more blocky, rectilinear boundaries.

  This difference arises because Euclidean distance is based on straight-line measurements, while Manhattan distance measures the distance along grid lines.

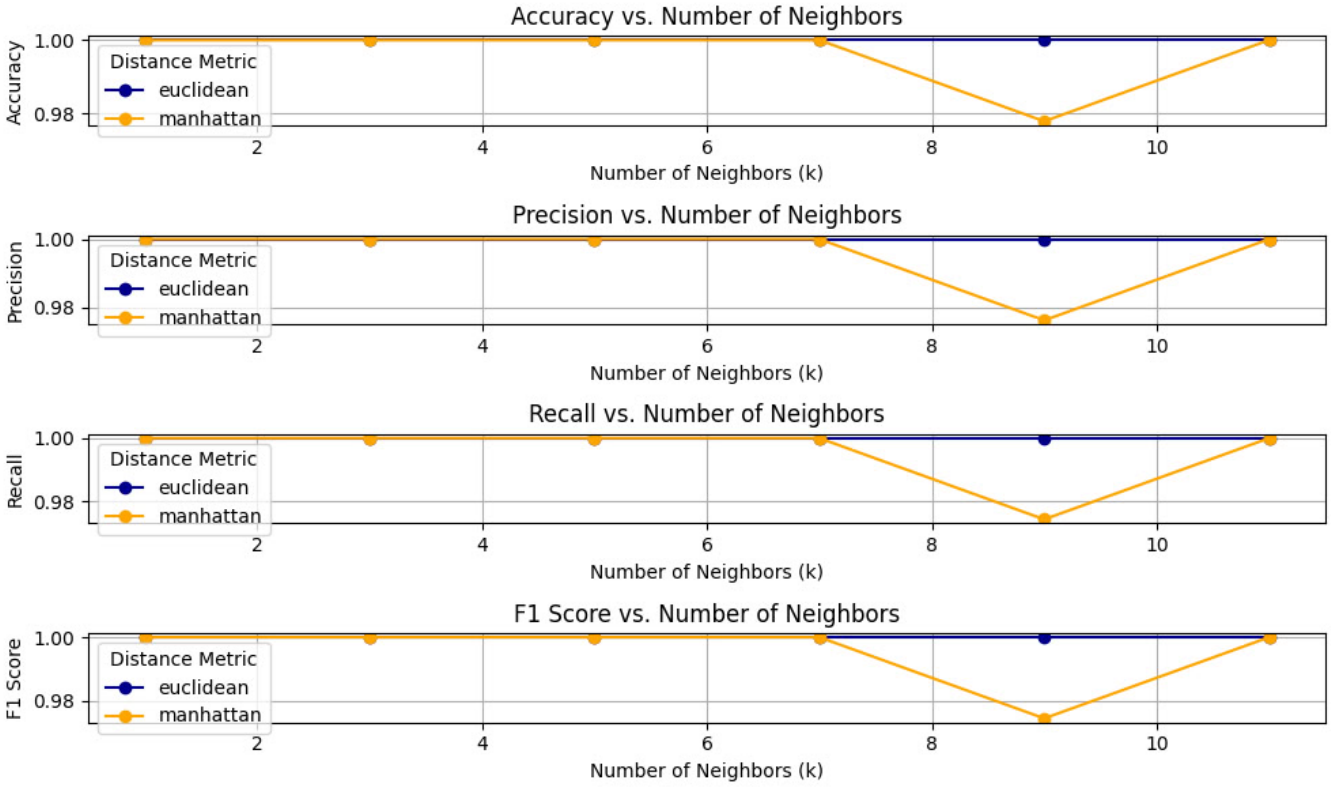
Misclassification Areas:

Areas where the background color does not match the majority of the points indicate regions where the classifier may misclassify some points.

For instance, in the middle region where Iris-versicolor (pink) and Iris-virginica (light brown) overlap, misclassifications are more likely.

This visualization helps in understanding how different values of k and the choice of distance metric affect the decision boundaries of the KNN classifier. It shows that smaller k values lead to more overfitting, while larger k values produce smoother, more generalized decision boundaries. The choice of distance metric also impacts the shape and smoothness of these boundaries, with Euclidean producing rounder and more intuitive separations, and Manhattan creating more grid-like, angular separations.

2. Visualization of performance metrics of a K-Nearest Neighbors (KNN) classifier



This graph displays four line plots showing how various performance metrics of a K-Nearest Neighbors (KNN) classifier change with different numbers of neighbors (k) and two different distance metrics (Euclidean and Manhattan). The four metrics evaluated are *Accuracy, Precision, Recall, and F1 Score.*

- ***Accuracy vs. Number of Neighbors (k)***
  For both distance metrics, the accuracy remains high (close to 1.0) for most values of k.
  A sharp drop in accuracy is seen with the Manhattan metric at k = 9, which then recovers at k = 11.
- ***Precision vs. Number of Neighbors (k)***
  Both distance metrics exhibit high precision for most values of k, but the Manhattan metric suffers a significant drop at k = 9.
- ***Recall vs. Number of Neighbors (k)***
  Both metrics exhibit very high recall, with a notable drop at k = 9 for the Manhattan distance metric.
- ***F1 Score vs. Number of Neighbors (k)***
  The Manhattan metric experiences a drop in F1 Score at k = 9, indicating a point where the model's balance of precision and recall deteriorates.
- ***Insight Gained***
  Consistency of Euclidean Metric: For all the metrics (Accuracy, Precision, Recall, F1 Score), the Euclidean distance metric maintains a consistently high performance across all values of k.

Instability of Manhattan Metric at k = 9: There's a noticeable drop in performance for the Manhattan distance metric specifically at k = 9, suggesting that for this particular value of k, the classifier struggles to maintain its predictive power.

General Behavior: As k increases, the performance generally remains stable, except for the Manhattan metric at k = 9, which is due to the specific nature of the dataset and how Manhattan distance calculates the nearest neighbors.

Problem #3: Perceptron Learning Algorithm

Source Code: **https://tinyurl.com/AI-Problems3**

Source Code Explanation:

- ***Splitting the Dataset:***

  The dataset is split into training and testing sets, with 70% of the data used for training and 30% for testing.

- ***Training:***

  A Perceptron model is initialized with *warm_start=True* to allow incremental training and track convergence.

  The Perceptron is trained over 200 epochs. During each epoch, it fits the model to the training data and records the number of misclassifications.

- ***Evaluation:***

  The model's performance is evaluated with accuracy, precision, recall, and F1-score on the test set, providing a comprehensive performance overview.

- ***Visualization:***

  Decision Boundaries: are visualized for the Perceptron model using a 2D projection of the feature space (Sepal Length and Sepal Width). This helps in understanding how the model separates different classes.

  Convergence: is plotted by showing the number of misclassifications over epochs. This illustrates how the model's performance improves (or not) over iterations.

  Additional Visualization: A pair plot of the entire dataset is created to visualize relationships between features and their separation by class.

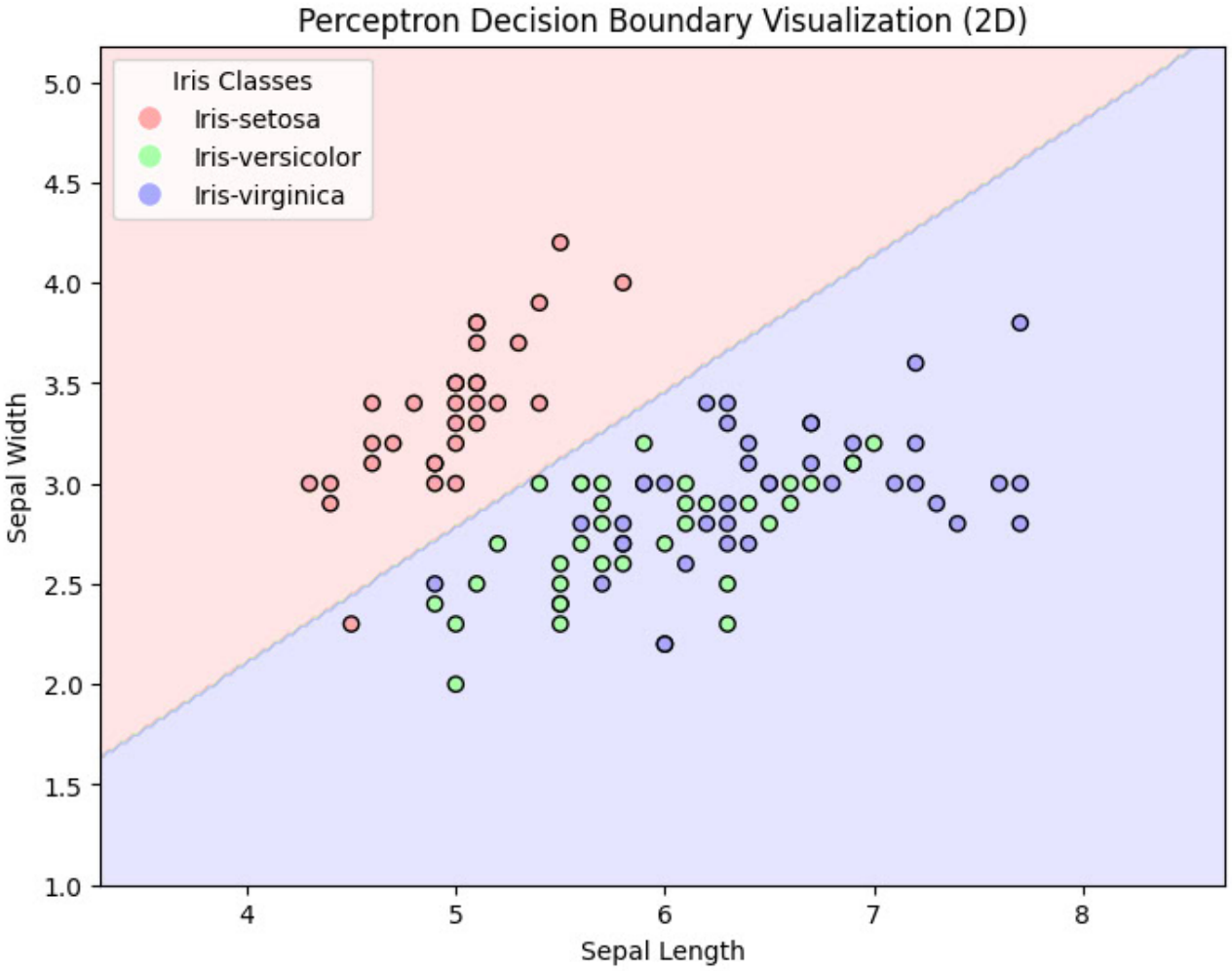
Output:

Perceptron Performance:

Accuracy: 0.76
Precision: 0.77
Recall: 0.72
F1 Score: 0.70

1. Visualization of Perceptron Decision Boundaries
2. Visualization of Perceptron Convergence

1. Visualization of Decision Boundaries



This plot shows how the Perceptron classifier has separated the data points in a 2D space based on Sepal Length and Sepal Width.

The ***background color*** (blue and green regions) represents the decision boundary where the classifier has divided the data into different classes.

The three ***classes of the Iris dataset*** (Iris-setosa, Iris-versicolor, and Iris-virginica) are represented by different color markers.

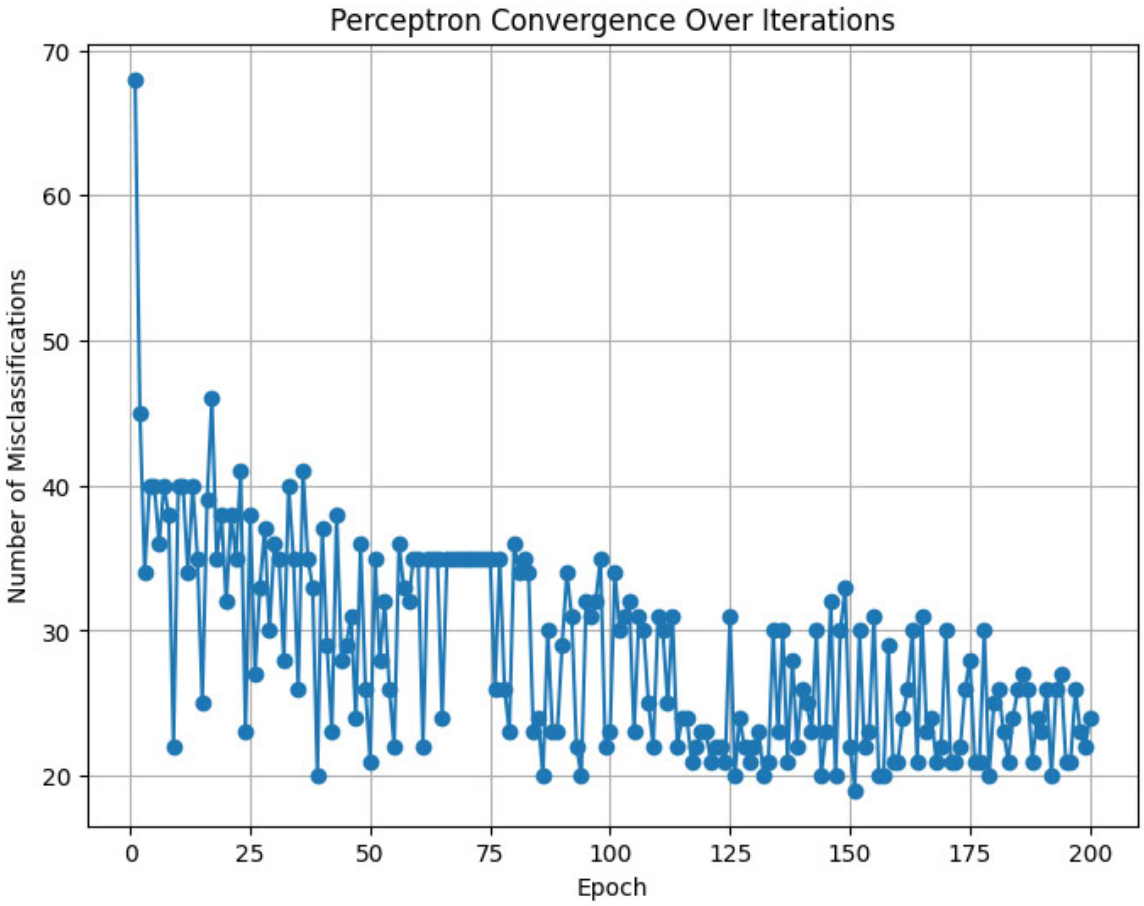
The ***decision boundary*** is a straight line, which is characteristic of the Perceptron model as it can only classify linearly separable data.

- ***Insight Gained***

  Points represent individual samples, and their position relative to the boundary indicates which class they are classified into.

  The graph shows how effectively the perceptron separates Iris-setosa from the other classes based on sepal dimensions. It visually demonstrates the linear separability of the classes in the feature space.

2. Visualization of Convergence



This ***plot*** tracks the number of misclassifications made by the Perceptron model over each epoch (iteration).

The ***y-axis*** represents the number of misclassifications, and the x-axis represents the number of epochs.

- ***Insight Gained***

  Initially, the number of misclassifications is high, indicating that the Perceptron model is not yet well-trained.

  As training progresses, the number of misclassifications decreases, showing that the model is learning and improving its classification accuracy over time.

  However, the plot shows some fluctuations, which suggests that the convergence process is not entirely smooth, possibly due to the inherent noise or complexity of the data.

  Towards the end, the number of misclassifications stabilizes, indicating that the model has converged.

Problem #4: Comparing Decision Tree, KNN, and Perceptron

Source Code: https://tinyurl.com/AI-Problems4

Source Code Explanation:

- ***Splitting the Dataset:***

  The dataset is split into training (70%) and testing (30%) sets using the *train_test_split()* function with fixed random_state for reproducibility.

- ***Training Classifiers:***

  Three classifiers are initialized: *DecisionTreeClassifier(), KNeighborsClassifier(), and Perceptron()*.

  Each classifier is trained on the training data using the *.fit()* method.
- ***Evaluating Models:***

  The models are evaluated using accuracy, precision, recall, and F1-score. The performance metrics for each classifier are stored in a dictionary (*results*) and converted into a DataFrame (*results_df*) for better visualization.
- ***Visualizing Decision Boundaries:***

  Decision boundaries are visualized using a custom function *plot_boundaries()*. This function plots the decision regions for each classifier on a 2D grid, using only the *SepalLength* and *SepalWidth* features for simplicity.

  The plots are generated for all three classifiers using *contourf* for decision regions and *scatter* for data points.
- ***Plotting Performance Metrics:***

  The performance metrics (accuracy, precision, recall, and F1-score) for each classifier are plotted as bar charts. The code uses *matplotlib.pyplot.bar()* to generate these plots, with colors distinguishing between the classifiers.

Decision Tree: provides interpretable results with its depth indicating model complexity.

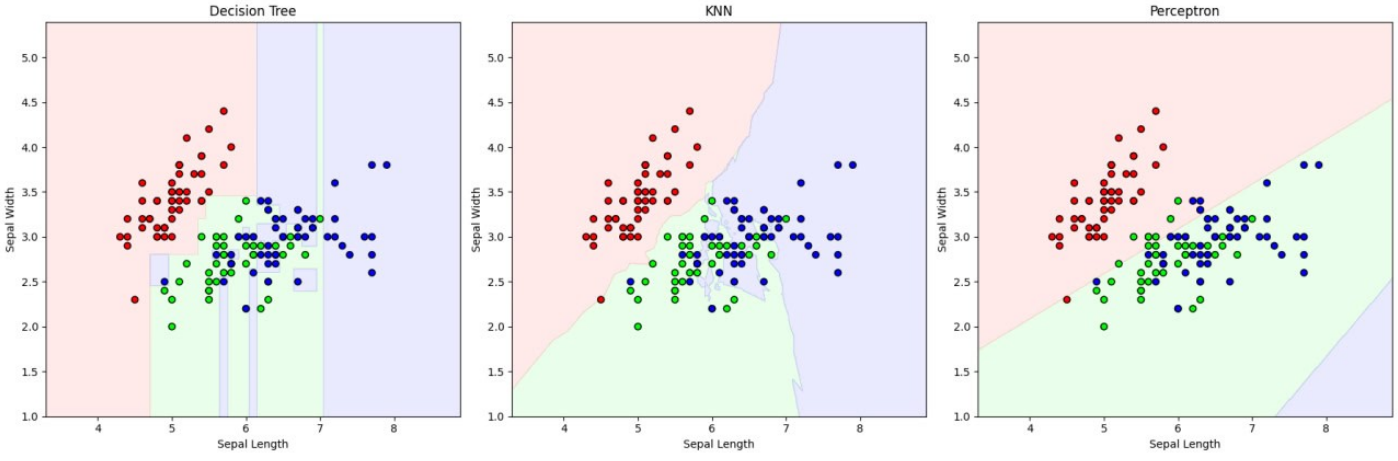
KNN: perform well with varying numbers of neighbors but can be computationally expensive.

Perceptron: As a simple linear model, does not perform as well on this dataset unless the classes are linearly separable.

Output: ***Model Performance:***

	Classifier	*Accuracy*	*Precision*	*Recall*	*F1 Score*
0	Decision Tree	1.0	1.000000	1.000000	1.000000
1	KNN	1.0	1.000000	1.000000	1.000000
2	Perceptron	0.8	0.863636	0.769231	0.737815

1. Visualization of the decision boundaries for all three classifiers



This image consists of three subplots, each showing the decision boundaries created by the Decision Tree, KNN, and Perceptron classifiers.

- ***Insight Gained:***

  Decision Tree (Left Plot):

  The regions are quite segmented and irregular, which indicates that the Decision Tree is capturing very specific details in the dataset, potentially leading to overfitting.

  Different colors represent different classes, and data points are plotted according to their Sepal Length and Sepal Width.

  Note: The depicted decision boundary is based on depth 12

  KNN (K-Nearest Neighbors) (Middle Plot):

  The boundaries are smoother and more continuous compared to the Decision Tree, reflecting KNN's reliance on local neighborhoods rather than strict rules.

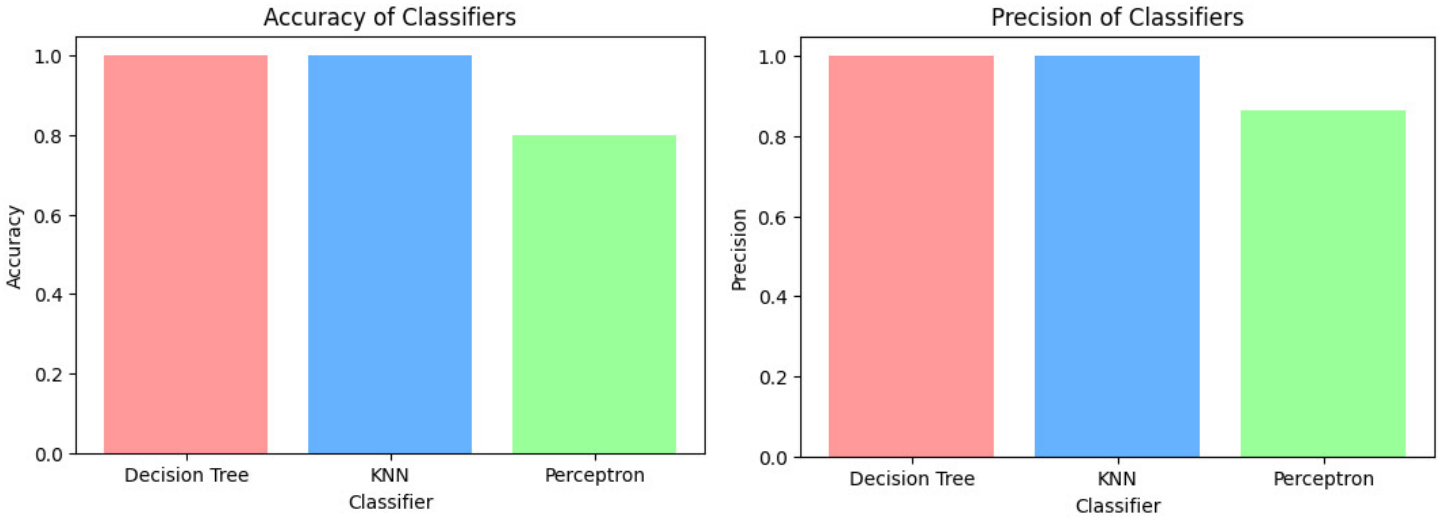
  The regions are less jagged, indicating a model that might be better at generalizing.

  Perceptron (Right Plot):

  The boundaries are linear, as Perceptron is a linear classifier, which can be limiting in scenarios where the classes are not linearly separable.

  The result is that the Perceptron fails to correctly classify all points, particularly in more complex regions of the dataset.

2. Visualization of the performance metrics for each classifier

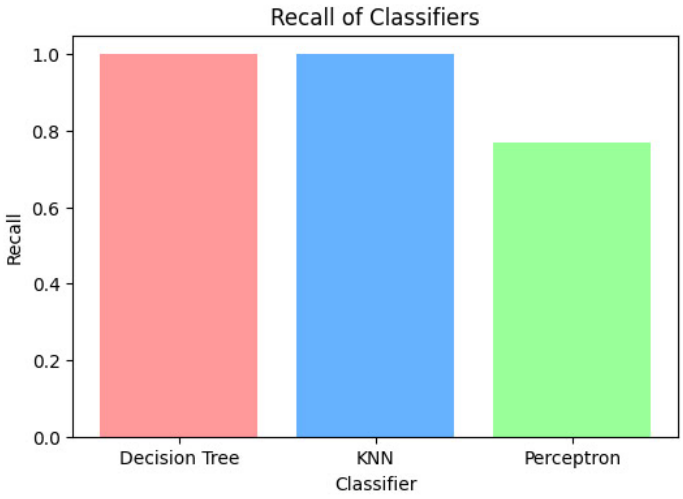


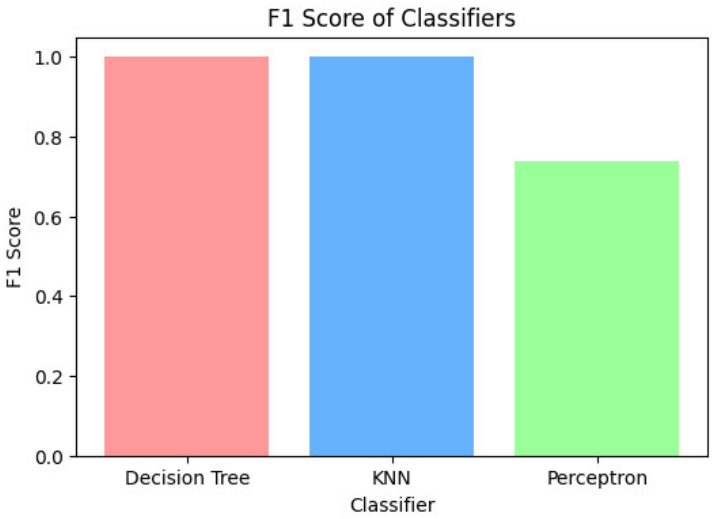
Accuracy: This chart shows that the Decision Tree and KNN classifiers achieved near-perfect accuracy, while the Perceptron had lower accuracy.

Accuracy reflects the proportion of correctly classified instances out of the total instances.

Precision: Precision is high for all classifiers but slightly lower for Perceptron.

Precision measures the proportion of true positive predictions among all positive predictions made by the classifier.





Recall: Both Decision Tree and KNN again score highly on recall, while the Perceptron performs worse.

Recall measures the proportion of actual positives correctly identified by the classifier.

F1 Score: The F1 Score, which is the harmonic mean of precision and recall, is highest for the Decision Tree and KNN, and lower for the Perceptron.

This indicates that Perceptron struggles more with this dataset compared to the other two models.

- ***Insight Gained***

  The ***Decision Tree*** and ***KNN*** classifiers are performing better overall, as seen by the high values in accuracy, precision, recall, and F1 score, and their decision boundaries are more aligned with the data.

  The ***Perceptron*** classifier, being a simple linear model, is less effective, particularly on this dataset where the decision boundaries are not linear.

Problem #5: K-Means Clustering and Visualization

Source Code: **https://tinyurl.com/AI-Problems5**

Source Code Explanation:

- ***Normalization:***

  The code normalizes the Iris dataset using *StandardScaler*, which ensures that the features have mean 0 and variance 1.

- ***K-Means Clustering:***

  K-Means clustering is correctly applied using *KMeans(n_clusters=3)* with random state 42.

- ***Visualization:***

  2D Plot: The code uses PCA to reduce the dataset to 2D and creates a scatter plot of the clusters with centroids marked in red.

  3D Plot: A 3D scatter plot is created using the original feature space to visualize clusters and centroids using the first three features *(SepalLength, SepalWidth, PetalLength)*.

- ***Cluster Comparison:***

  The code maps clusters to actual species based on the majority class in each cluster. It then calculates the confusion matrix and accuracy score to compare the clustering results with the true labels. This approach effectively evaluates the clustering performance.

- ***Silhouette Scores:***

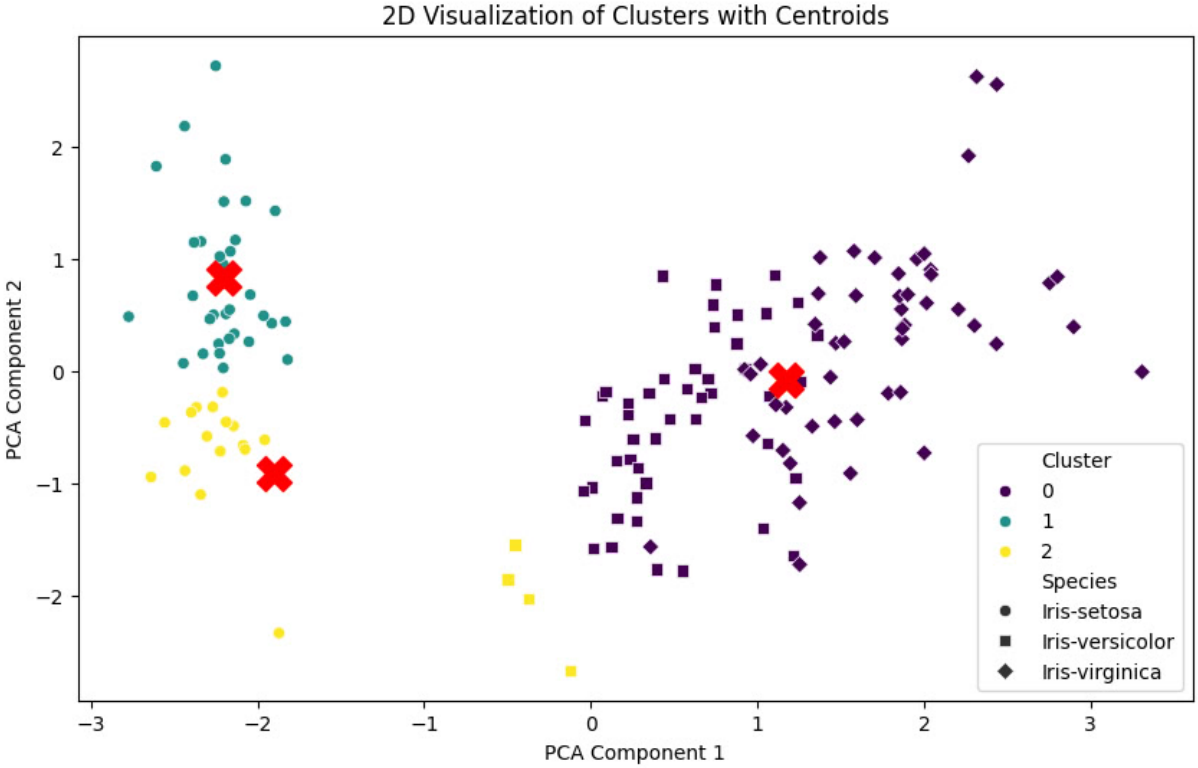
  Silhouette scores are calculated for both individual data points and as an overall average, and the results are plotted to visualize the distribution of silhouette scores across the clusters.

  The silhouette plot includes vertical line to indicate average silhouette score, which provides an intuitive way to evaluate clustering quality.

Output: ***Accuracy Score:*** 0.6666666666666666

Average Silhouette Score: 0.4787241921049546

1. 2D Visualization of Clusters with Centroids

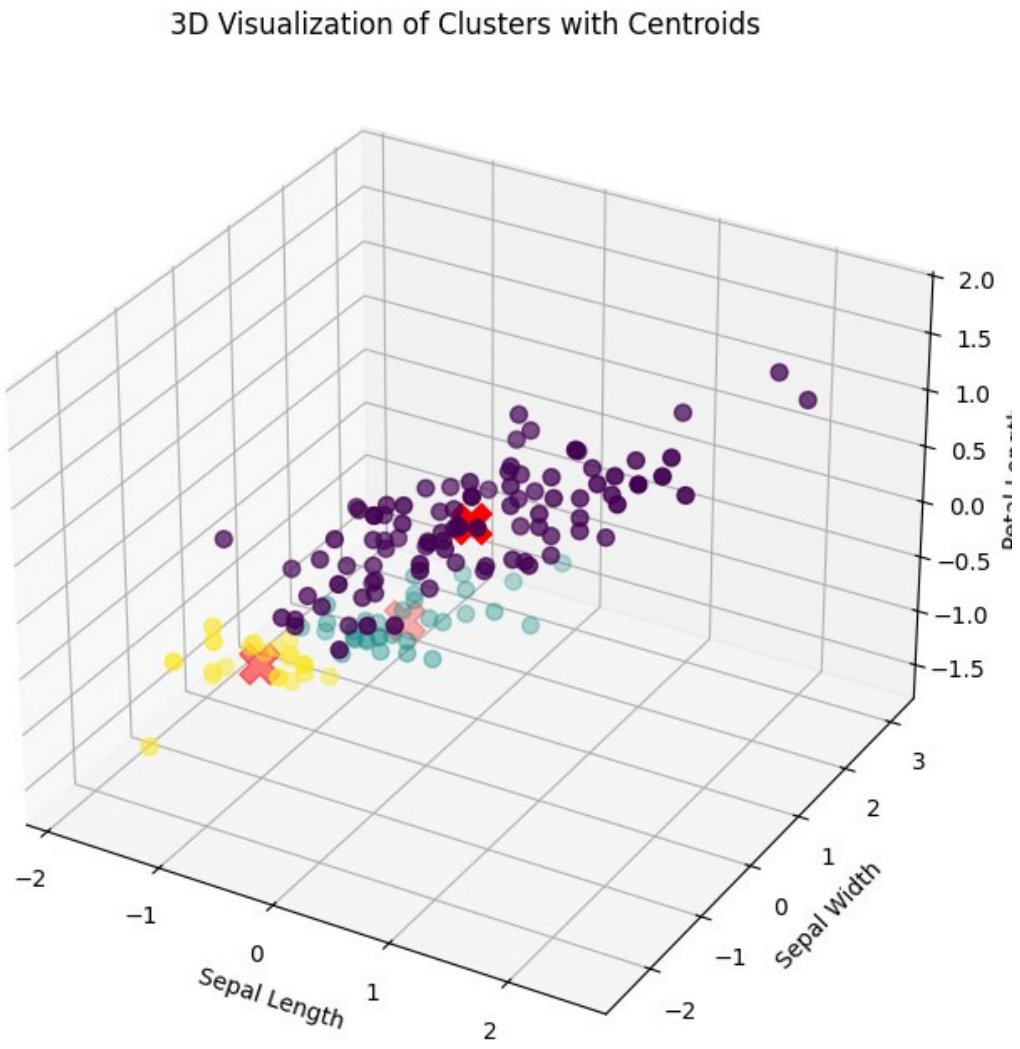


This scatter plot shows the results of k-means clustering on the Iris dataset, reduced to two dimensions using PCA. Each point represents an iris sample, colored according to its cluster. The red crosses indicate the centroids of each cluster.

- ***Insight Gained:***

  This graph helps to visually assess how well the data points are grouped into clusters in a reduced two-dimensional space. The proximity of points within the same cluster and the separation between different clusters can indicate how distinct the clusters are. It also shows how closely the clusters align with the actual species labels.

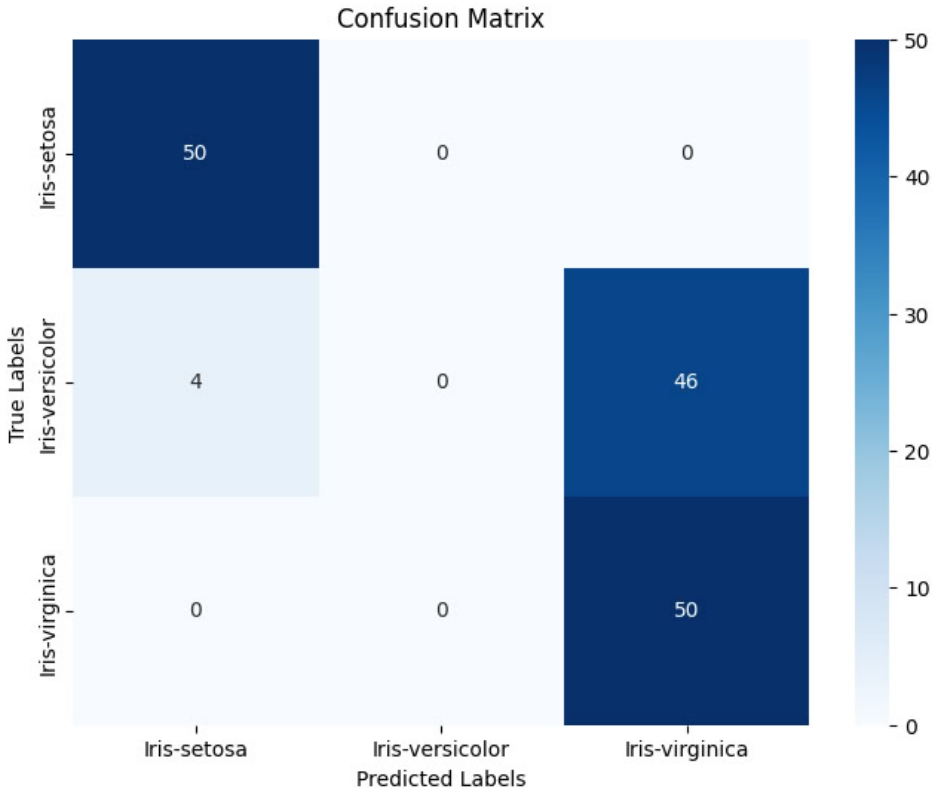
2. 3D Visualization of Clusters with Centroids



This 3D scatter plot visualizes the clustering using three features of the dataset. Each point is colored by its cluster assignment, and the red crosses represent the centroids.
This provides a more detailed view of how data points are grouped in higher dimensions.

- ***Axes:*** ***X-axis:*** Sepal Length
  Y-Axis: Petal Length
  Z;Axis: Sepal Width
- ***Insight Gained:***
  By adding an extra dimension, this plot provides a more nuanced understanding of the clustering. It can reveal overlapping clusters or outliers that may not be as apparent in 2D.
  This can help in evaluating the effectiveness of the clustering when more features are considered.

3. Confusion Matrix

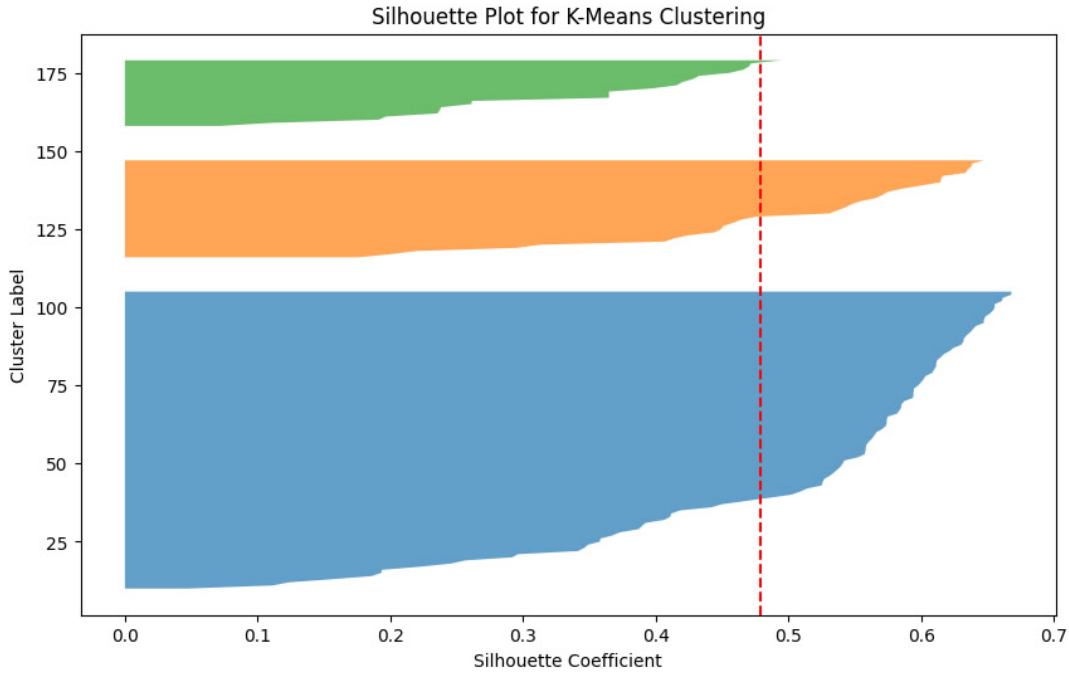


This heatmap compares the predicted cluster assignments to the actual species labels. It shows how well the clustering aligns with the true classifications of the Iris dataset. The diagonal elements indicate correct predictions.

- ***Insight Gained:***

  It quantitatively evaluates the clustering accuracy by comparing predicted clusters to actual species. The matrix shows which species are most often confused with each other, highlighting potential weaknesses in the clustering algorithm, such as misclassifications between similar species.

4. Silhouette Plot for K-Means Clustering



This plot shows the silhouette coefficients for each sample in the dataset. It visualizes how well-separated the clusters are, with a vertical red dashed line indicating the average silhouette score. Higher values suggest better-defined clusters.

- ***Insight Gained:***

  This plot provides a measure of how well-separated the clusters are. A higher silhouette score indicates better-defined clusters. The plot also reveals the consistency within clusters, helping to identify clusters that might be too broad or too tight.

Problem #6: K-Medoids Clustering and Comparison with K-Means

Source Code: https://tinyurl.com/AI-Problems6

Source Code Explanation:

- ***Normalization of the Iris Dataset***

  The code normalizes the Iris dataset using *StandardScaler* from *sklearn*. This ensures that all features (*Sepal Length, Sepal Width, Petal Length, Petal Width*) are scaled to have 0 mean and 1 variance, which is crucial for clustering algorithms to function correctly.

- ***K-Medoids Clustering***

  The code implements a custom K-Medoids clustering algorithm. It initializes medoids randomly, assigns data points to the nearest medoid, and updates medoids based on minimizing the sum of distances within clusters. It stops when medoids no longer change or the maximum number of iterations is reached.

- ***Visualization in 2D and 3D***

  The code generates visualizations for both K-Medoids and K-Means clustering results:

  2D Plot: Displays *Sepal Length vs. Sepal Width* with clusters color-coded and medoids/centroids marked.

  3D Plot: Shows *Sepal Length, Sepal Width, and Petal Length,* with clusters color-coded and medoids/centroids highlighted.

- ***Comparison with Actual Class Labels***

  The code calculates various metrics to compare clustering results with actual class labels:

  Confusion Matrix: Shows how well the clusters align with true classes.

  Purity: Measures the fraction of correctly classified points in each cluster.

  Accuracy: Provides an additional comparison metric (less typical for unsupervised learning).

- ***Performance Comparison of K-Means and K-Medoids***

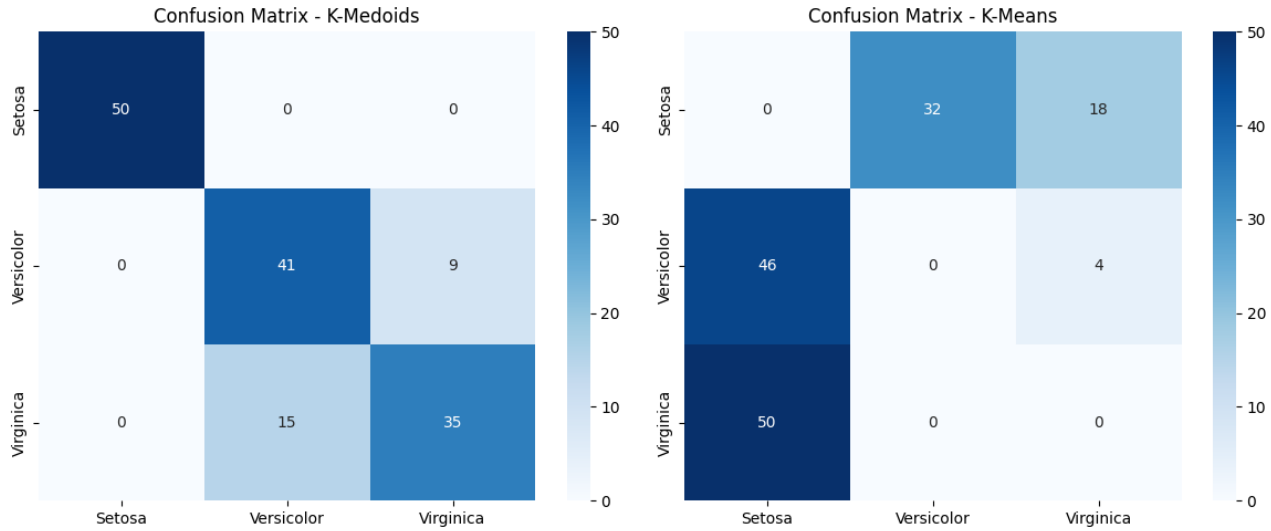
  The code evaluates clustering performance using:

  Silhouette Score: Measures how similar points are to their own cluster compared to other clusters.

  Adjusted Rand Index (ARI): Compares the similarity between true labels and cluster assignments.

Output: *Convergence reached after 5 iterations.*
Silhouette Score for K-Medoids: 0.4593
Silhouette Score for K-Means: 0.4787
Adjusted Rand Index for K-Medoids: 0.6676
Adjusted Rand Index for K-Means: 0.4290
Purity for K-Medoids: 0.8600
Purity for K-Means: 0.6667

1. Visualization of Confusion Matrix



- ***K-Medoids Confusion Matrix (Left)***

  The confusion matrix for the K-Medoids clustering algorithm shows how well the clustering algorithm assigned data points to clusters compared to the actual labels of the Iris dataset.

  The matrix displays three clusters (0, 1, 2) corresponding to the three Iris species. The diagonal elements represent correct cluster assignments (true positives), while off-diagonal elements indicate incorrect assignments (misclassifications).

  For example, all instances of class 0 were misclassified into class 1, indicating that K-Medoids failed to identify this class correctly

- ***K-Means Confusion Matrix (Right)***

  Similar to the K-Medoids confusion matrix, this one shows the performance of the K-Means clustering algorithm.

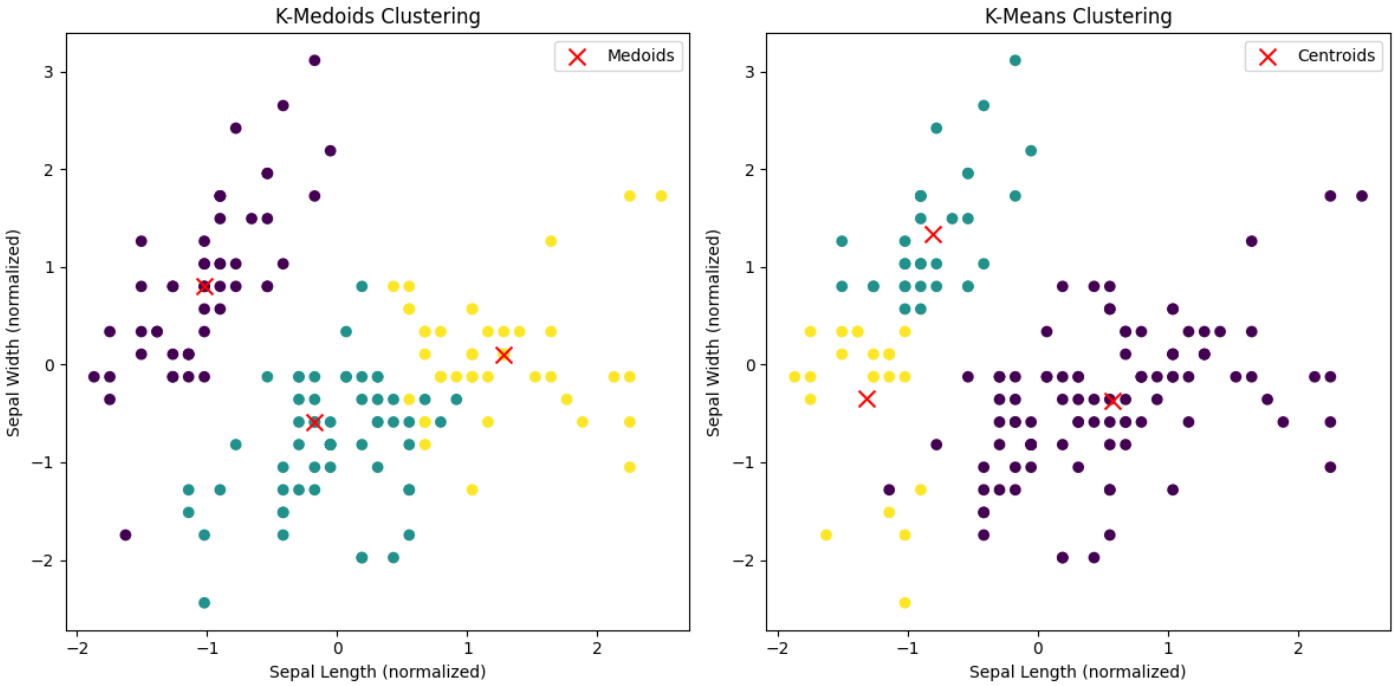
  The matrix shows that K-Means also misclassified all instances of class 0 into class 1, but compared to K-Medoids, it managed to distribute the data more evenly across other classes

- ***Insight Gained***

  K-Medoids: Perfectly clusters "*Setosa*."
  Some overlap between "*Versicolor*" and "*Virginica*."
  Indicates better separation for Setosa but not with other two classes.
  K-Means: Misclassifies some "*Setosa*" as "*Versicolor*" and vice versa.
  Shows significant overlap between "*Versicolor*" and "*Virginica*."
  Suggests that K-Means struggles with distinguishing between these classes.

2. 2D Visualization of Clustering

•



- ***K-Medoids Clustering (Left)***

  This plot shows the 2D clustering result of the K-Medoids algorithm, where the Sepal Length and Sepal Width features (normalized) are plotted on the x and y axes, respectively.

  Data points are colored according to their assigned clusters, and red "X" marks indicate the medoids (representative points for each cluster).

  This plot visualizes the separation of data points into clusters based on their sepal features, with the medoids acting as central points around which clusters are formed.

- ***K-Means Clustering (Right)***

  This plot shows the 2D clustering result of the K-Means algorithm on the same features (Sepal Length and Sepal Width).

  Similar to the K-Medoids plot, data points are colored by their cluster assignment, and blue dots represent the centroids (mean points of the clusters).

- ***Insight Gained***

  The plot highlights how K-Means clusters the data and where the centroids are located, which differ from the medoids in the K-Medoids plot.

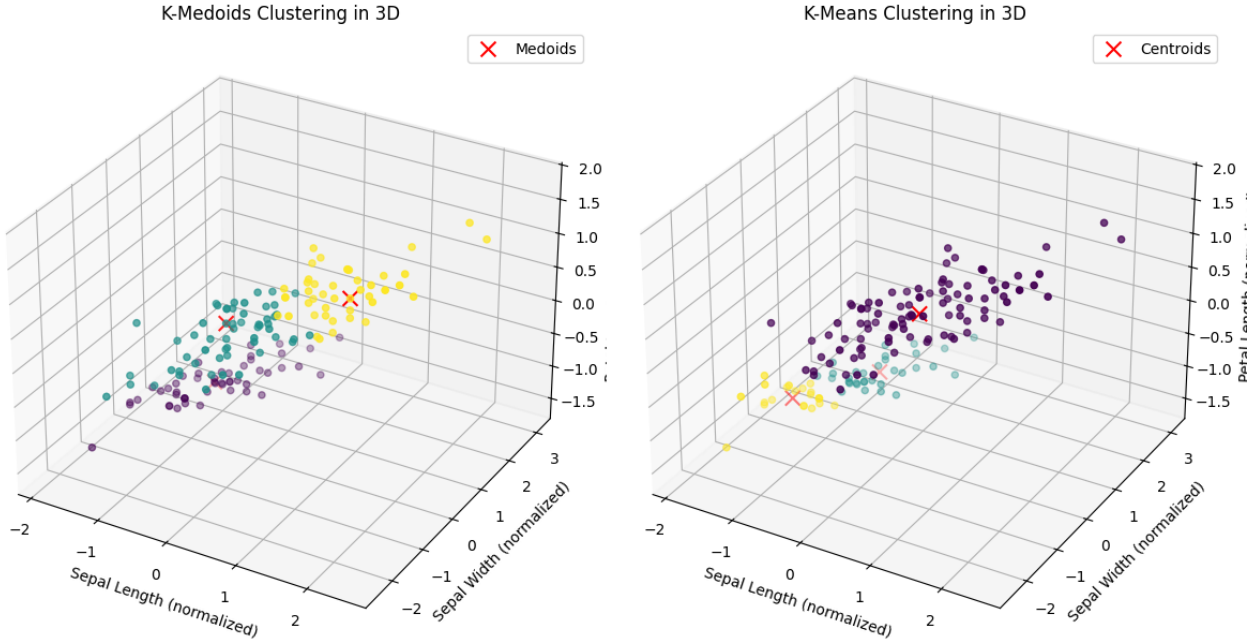
  K-Medoids: Clusters appear more compact and medoids are central.

  Clear separation for some clusters, but overlap exists.

  K-Means: Clusters are more spread out.

  Centroids might not be as representative of the cluster centers due to outliers.

3. 3D Visualization of Clustering



This graph depicts 3D clustering results for K-Medoids (left) and K-Means (right) on the Iris dataset. The clusters are shown along three normalized feature dimensions: *Sepal Length, Sepal Width, and Petal Length.*

K-Medoids (left): Data points are clustered around three medoids (shown as red crosses), with each color representing a different cluster.

K-Means (right): Data points are clustered around centroids (shown as red crosses), with each color also representing a different cluster.

The 3D visualization provides a more detailed view of how the clusters are distributed in space. Both algorithms show overlap between clusters, with K-Medoids clusters appearing less symmetrical than K-Means, due to the algorithm's reliance on medoids instead of calculated centroids.

- ***Insight Gained***

  K-Medoids: Provides a better view of cluster separation in 3D.
  Shows how medoids effectively represent the center of each cluster.

  K-Means: Highlights overlap among clusters in 3 dimensions.
  Centroids might be skewed by the distribution of points.

Cluster Overlap: Both K-Medoids and K-Means exhibit significant overlap between clusters, suggesting that the features (*Sepal Length, Sepal Width, and Petal Length*) might not perfectly separate the Iris classes, indicating that additional features may be needed for better separation.

K-Medoids vs. K-Means Shape: The clusters in K-Medoids appear more irregular in shape, which is expected as it assigns data points to the nearest medoid. In contrast, K-Means clusters are more spherical, reflecting the influence of centroids calculated as the mean of data points in each cluster.

Centroid and Medoid Placement: The placement of centroids (K-Means) and medoids (K-Medoids) influences the cluster shapes. K-Means, due to its mean-based approach, can adapt to the distribution of points more symmetrically than K-Medoids, which selects actual data points as cluster centers.