Computer Laboratory I - Machine Learning Lab Manual

Assignment 1

Dimensionality Reduction using PCA Algorithm for Wine Classification

Introduction:

Principal Component Analysis (PCA) is a powerful technique used for dimensionality reduction by transforming high-dimensional data into a lower-dimensional space while retaining as much variance as possible.

Objective:

The objective of this lab is to utilize the PCA algorithm to reduce the dimensionality of the wine dataset while preserving the most significant variations in the data. The aim is to simplify the dataset representation and improve the distinction between red and white wines using the transformed principal components.

Dataset: Wine.csv

Theory-

Step 1: Importing Libraries and Loading the Dataset

1. Open your preferred Python environment (such as Jupyter Notebook).

2. Import the necessary libraries: pandas, numpy, matplotlib.pyplot, and sklearn.

3. Load the dataset using pandas from the provided CSV link.

Step 2: Data Preprocessing

1. Check the basic information about the dataset using the `info()` function.

2. Separate the features (variables) and the target variable (wine type) from the dataset.

3. Standardize the features to have zero mean and unit variance using the `StandardScaler` from sklearn.

Step 3: Applying PCA Algorithm

1. Import the `PCA` class from sklearn.

2. Create an instance of the `PCA` class.

3. Fit the PCA model to the standardized data using the `fit()` method.

4. Explore the explained variance ratio for each principal component using the `explained_variance_ratio_` attribute.

5. Plot a cumulative explained variance plot to visualize how much variance is captured by a certain number of principal components.

Step 4: Choosing the Number of Principal Components

1. Based on the cumulative explained variance plot, decide how many principal components to retain for dimensionality reduction.

2. Retrain the PCA model with the selected number of components.

Step 5: Transforming Data and Visualization

1. Transform the standardized data using the `transform()` method of the PCA model.

2. Create a new DataFrame with the transformed data and assign appropriate column names.

3. Visualize the transformed data using scatter plots, with the principal components as axes.

Step 6: Wine Classification using Principal Components

1. Split the transformed data and the target variable (wine type) into training and testing sets.

2. Choose a classification algorithm (e.g., Logistic Regression, Random Forest) and train it using the transformed data.

3. Evaluate the classification model's performance on the testing set.

Conclusion:

In this lab, we have successfully applied the PCA algorithm to the wine dataset, reducing its dimensionality while retaining significant variations in the data. By visualizing the transformed data and applying a classification algorithm, we have demonstrated how the reduced set of principal components can help distinguish between red and white wines more effectively than the original high-dimensional data. This technique showcases the power of dimensionality reduction in simplifying complex datasets while preserving meaningful information.

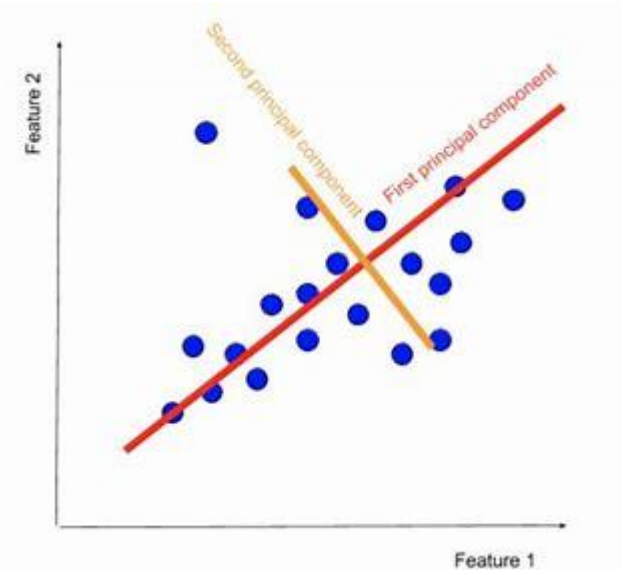


Fig 1 Principal Component Analysis

Assignment 2

Regression Analysis for Predicting Uber Ride Prices

Introduction:

Regression analysis is a statistical technique used to model the relationship between a dependent variable and one or more independent variables. In this analysis, we have predict the price of an Uber ride based on various features such as pickup point, drop-off location, etc.

Dataset:

Dataset :Uber Fares Dataset

Analysis Steps:

Step 1: Importing Libraries and Loading the Dataset

1. Open your preferred Python environment (e.g., Jupyter Notebook).

2. Import the necessary libraries: pandas, numpy, matplotlib.pyplot, seaborn, sklearn.

3. Load the dataset using pandas from the provided CSV link.

Step 2: Data Preprocessing

1. Check basic information about the dataset using the `info()` and `head()` functions.

2. Handle missing values if any.

3. Convert categorical variables into numerical representations using techniques like one-hot encoding.

4. Normalize or standardize the numerical features if needed.

Step 3: Identifying Outliers

1. Visualize the distribution of numerical variables using box plots or histograms.

2. Identify potential outliers using appropriate techniques (e.g., IQR, Z-score).

Step 4: Checking Correlations

1. Calculate the correlation matrix of numerical features.

2. Visualize the correlation matrix using a heatmap to identify strong correlations.

Step 5: Implementing Regression Models

1. Separate the features (independent variables) and the target variable (ride price) from the dataset.

2. Split the data into training and testing sets using `train_test_split` from sklearn.

3. Implement linear regression, ridge regression, and Lasso regression models using the appropriate classes from sklearn.

4. Fit each model to the training data.

Step 6: Evaluating Models

1. Make predictions using each model on the testing set.

2. Calculate metrics such as R-squared, Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), etc., for each model to evaluate their performance.

3. Compare the performance metrics of the models to determine which one performs better.

Step 7: Model Comparison

1. Compare the performance of linear regression, ridge regression, and Lasso regression models.

2. Choose the model that yields the best performance based on the evaluation metrics.

3. Conclude the analysis by discussing the insights gained, model performance, and potential areas for improvement.

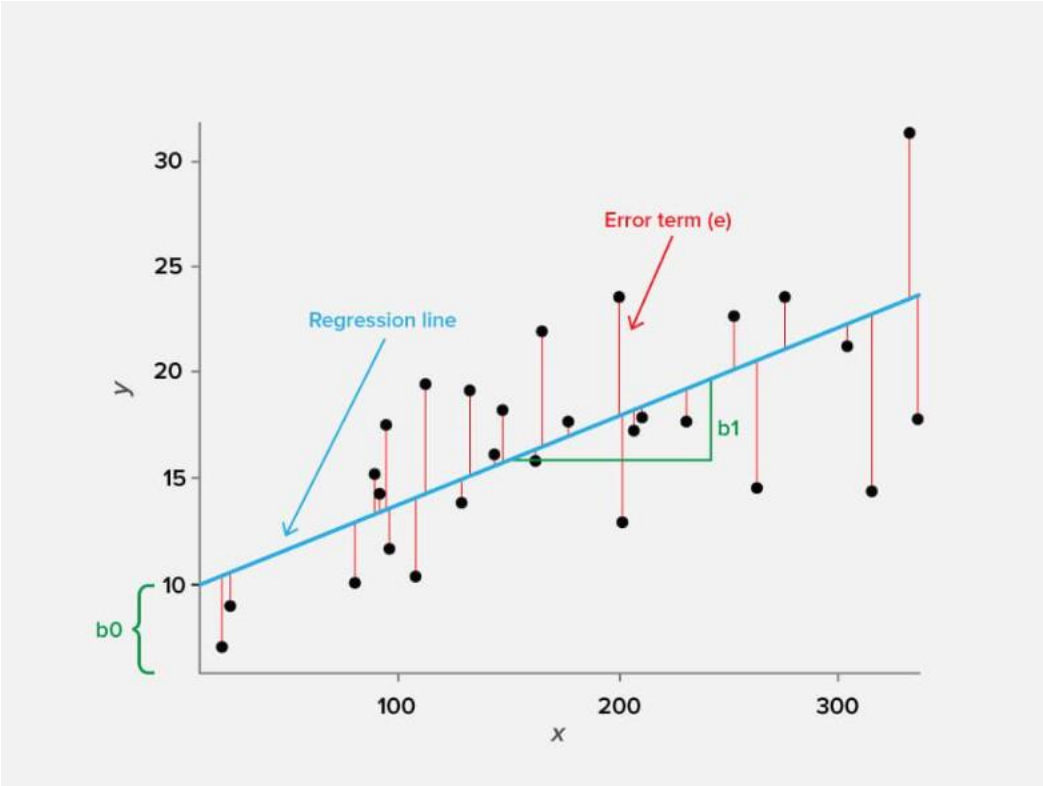


Fig 2 Regression Model

Conclusion:

In this analysis, you successfully performed regression analysis to predict the price of Uber rides based on various features. By comparing the metrics of these models, we gained insights into which model provides the best predictions for ride prices. This analysis showcases the power of regression techniques in predicting real-world outcomes based on relevant features.

Assignment 3

Classification Analysis using Support Vector Machines (SVM) for Handwritten Digit Classification

Introduction:

Support Vector Machines (SVM) is a powerful machine learning algorithm used for classification tasks. In this analysis, you will implement SVM to classify images of handwritten digits into their respective numerical classes (0 to 9). The popular MNIST dataset, which consists of a large number of labelled handwritten digit images, will be used for this task.

Dataset: MNIST dataset

Analysis Steps:

Step 1: Importing Libraries and Loading the Dataset

1. Open your preferred Python environment (e.g., Jupyter Notebook).

2. Import the necessary libraries: numpy, matplotlib.pyplot, scikit-learn (for SVM and dataset).

3. Load the MNIST dataset using the appropriate function from scikit-learn.

Step 2: Data Preprocessing

1. Check basic information about the dataset using the `info()` and `shape` attributes.

2. Visualize a few sample images along with their labels using matplotlib.

Step 3: Data Splitting

1. Separate the features (pixel values) and the target variable (digit labels) from the dataset.

2. Split the data into training and testing sets using `train_test_split` from scikit-learn.

Step 4: SVM Model Implementation

1. Import the `SVC` class from scikit-learn.

2. Create an instance of the `SVC` class with appropriate hyperparameters (kernel, C value, etc.).

3. Fit the SVM model to the training data using the `fit()` method.

Step 5: Model Evaluation

1. Make predictions on the testing set using the trained SVM model.

2. Calculate metrics such as accuracy, precision, recall, and F1-score using functions from scikit-learn's `metrics` module.

3. Visualize a confusion matrix to understand the classification performance in more detail.

Step 6: Hyperparameter Tuning

1. Experiment with different hyperparameter values (kernel type, C value, etc.) to find the best configuration.

2. Use techniques like grid search or randomized search for efficient hyperparameter tuning.

Step 7: Conclusion

1. Summarize the results obtained from the SVM classification.

2. Discuss the accuracy and performance of the SVM model in classifying handwritten digits.

3. Highlight any insights gained from misclassified images or challenging cases.

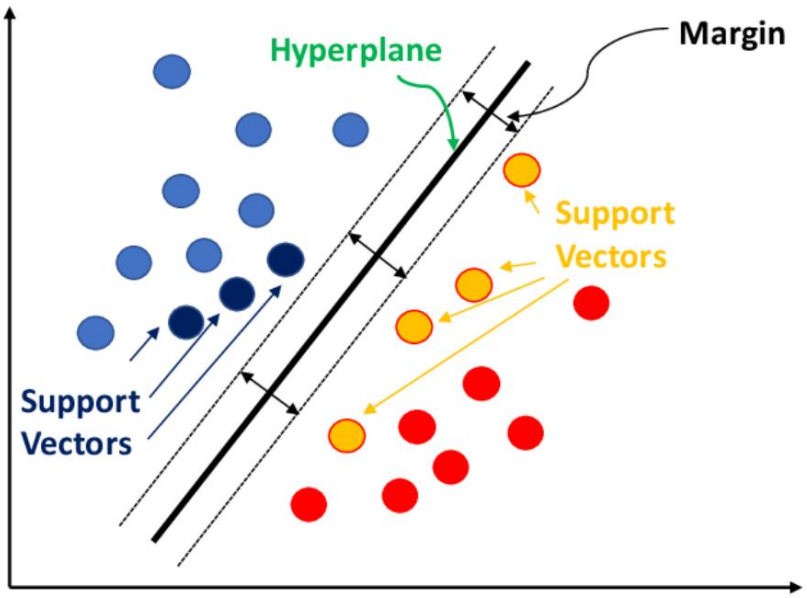


Fig 3 SVM

Conclusion:

In this analysis, we successfully implemented a Support Vector Machine (SVM) to classify images of handwritten digits into their respective numerical classes. By training and evaluating the SVM model on the MNIST dataset, you gained insights into its classification performance. This analysis demonstrates the effectiveness of SVMs in solving image classification tasks and showcases their potential in various real-world applications.

Assignment 4

Implement K-Means clustering on Iris.csv dataset. Determine the number of clusters using the elbow method.

Introduction:

Clustering is an unsupervised machine learning technique used to group similar data points together. K-Means is a popular clustering algorithm that partitions data into a predefined number of clusters and to implement K-Means clustering on the Iris dataset to group similar flowers based on their features. Additionally, use the elbow method to determine the optimal number of clusters for the dataset.

Dataset: Iris dataset

Analysis Steps:

Step 1: Importing Libraries and Loading the Dataset

1. Open your preferred Python environment (e.g., Jupyter Notebook).

2. Import the necessary libraries: pandas, numpy, matplotlib.pyplot, seaborn, sklearn.

3. Load the Iris dataset using pandas from the provided CSV link.

Step 2: Data Preprocessing

1. Check basic information about the dataset using the `info()` and `head()` functions.

2. Separate the features (attributes) from the target variable (species).

Step 3: Determining the Optimal Number of Clusters (Elbow Method)

1. Import the `KMeans` class from sklearn.

2. Implement the K-Means algorithm for a range of cluster numbers (e.g., from 1 to 10).

3. For each cluster number, compute the sum of squared distances (inertia) of samples to their closest cluster center.

4. Visualize the inertia values using a plot to identify the "elbow point," which indicates the optimal number of clusters.

Step 4: Implementing K-Means Clustering

1. Choose the optimal number of clusters based on the elbow method.

2. Create an instance of the `KMeans` class with the chosen number of clusters.

3. Fit the K-Means model to the feature data using the `fit()` method.

Step 5: Visualizing Clusters

1. Assign cluster labels to each data point using the `labels_` attribute of the trained K-Means model.

2. Visualize the clusters using scatter plots based on two chosen features.

3. Mark the cluster centers on the scatter plot.

Step 6: Conclusion

1. Summarize the insights gained from the K-Means clustering.

2. Discuss how well the clusters represent the actual species labels.

3. Reflect on the use of the elbow method for determining the optimal number of clusters.

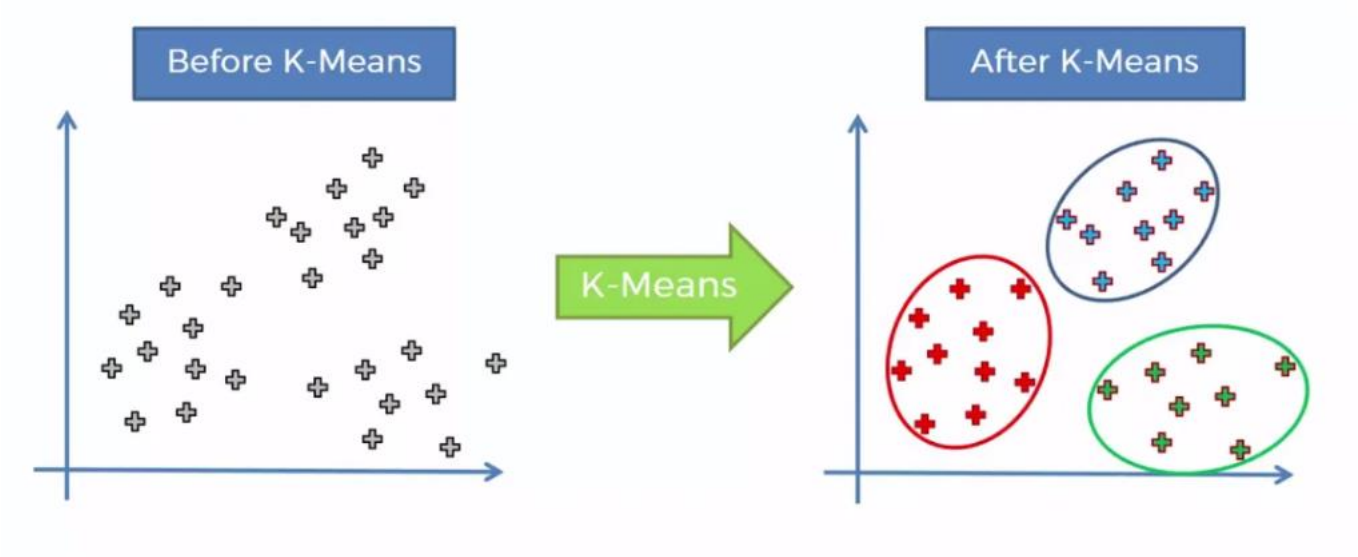


Fig 4 K-means Clustering

Conclusion:

In this analysis, we successfully implemented K-Means clustering on the Iris dataset. By determining the optimal number of clusters using the elbow method and visualizing the clusters, you gained insights into how the K-Means algorithm groups similar flowers together based on their features. This analysis highlights the effectiveness of K-Means clustering in discovering patterns and relationships within the data.

Assignment 5

Implement Random Forest Classifier model to predict the safety of the car.

Introduction:

Ensemble learning is a machine learning technique that combines multiple individual models to improve predictive performance and generalization. Random Forest is a popular ensemble method that builds multiple decision trees and combines their predictions to make more accurate and robust predictions and implement a Random Forest Classifier to predict the safety level of cars based on their attributes.

Dataset: Car Evaluation Dataset

Analysis Steps:

Step 1: Importing Libraries and Loading the Dataset

1. Open your preferred Python environment (e.g., Jupyter Notebook).

2. Import the necessary libraries: pandas, numpy, matplotlib.pyplot, seaborn, sklearn.

3. Load the car evaluation dataset using pandas from the provided CSV link.

Step 2: Data Preprocessing

1. Check basic information about the dataset using the `info()` and `head()` functions.

2. Handle any missing values in the dataset.

3. Convert categorical variables into numerical representations using techniques like label encoding or one-hot encoding.

Step 3: Data Splitting

1. Separate the features (attributes) and the target variable (safety level) from the dataset.

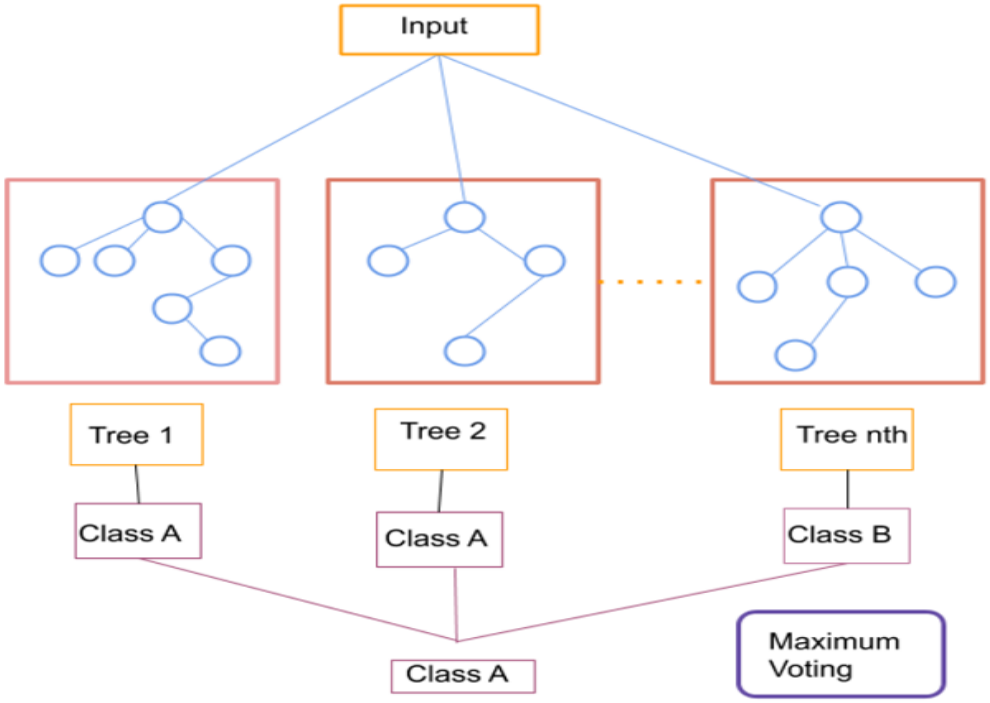
2. Split the data into training and testing sets using `train_test_split` from sklearn.

Step 4: Implementing Random Forest Classifier

1. Import the `RandomForestClassifier` class from sklearn.

2. Create an instance of the `RandomForestClassifier` class with appropriate hyperparameters.

3. Fit the Random Forest model to the training data using the `fit()` method.



Step 5: Model Evaluation

1. Make predictions on the testing set using the trained Random Forest model.

2. Calculate metrics such as accuracy, precision, recall, and F1-score using functions from scikit-learn's `metrics` module.

3. Visualize a confusion matrix to understand the classification performance in more detail.

Step 6: Feature Importance

1. Extract feature importances from the trained Random Forest model.

2. Visualize the importance of each feature using a bar plot.

Step 7: Conclusion

1. Summarize the results obtained from the Random Forest classification.

2. Discuss the accuracy and performance of the model in predicting car safety levels.

3. Highlight the most important features that contribute to the model's predictions.

Conclusion:

In this analysis, we successfully implemented a Random Forest Classifier to predict car safety levels based on the attributes of cars. By training and evaluating the model on the car evaluation dataset, we gained insights into its classification performance and the importance of different features in making predictions. This analysis demonstrates the effectiveness of ensemble methods like Random Forest in making accurate predictions in complex classification tasks.

Assignment 6

Reinforcement Learning **:** Implement Reinforcement Learning using an example of a maze environment that the agent needs to explore.

Introduction:

Reinforcement Learning is a branch of machine learning that deals with decision-making in an environment to maximize a cumulative reward. Q-Learning is a popular algorithm within reinforcement learning that enables an agent to learn an optimal policy for making decisions in an environment through exploration and exploitation and implement Q-Learning to solve a maze environment, where the agent learns to navigate through the maze to reach a goal while avoiding obstacles.

Analysis Steps:

Step 1: Environment Setup

1. Define the maze environment as a grid with walls, start position, and goal position.

2. Define the actions that the agent can take (e.g., up, down, left, right).

3. Set the rewards for different scenarios (e.g., reaching the goal, hitting a wall).

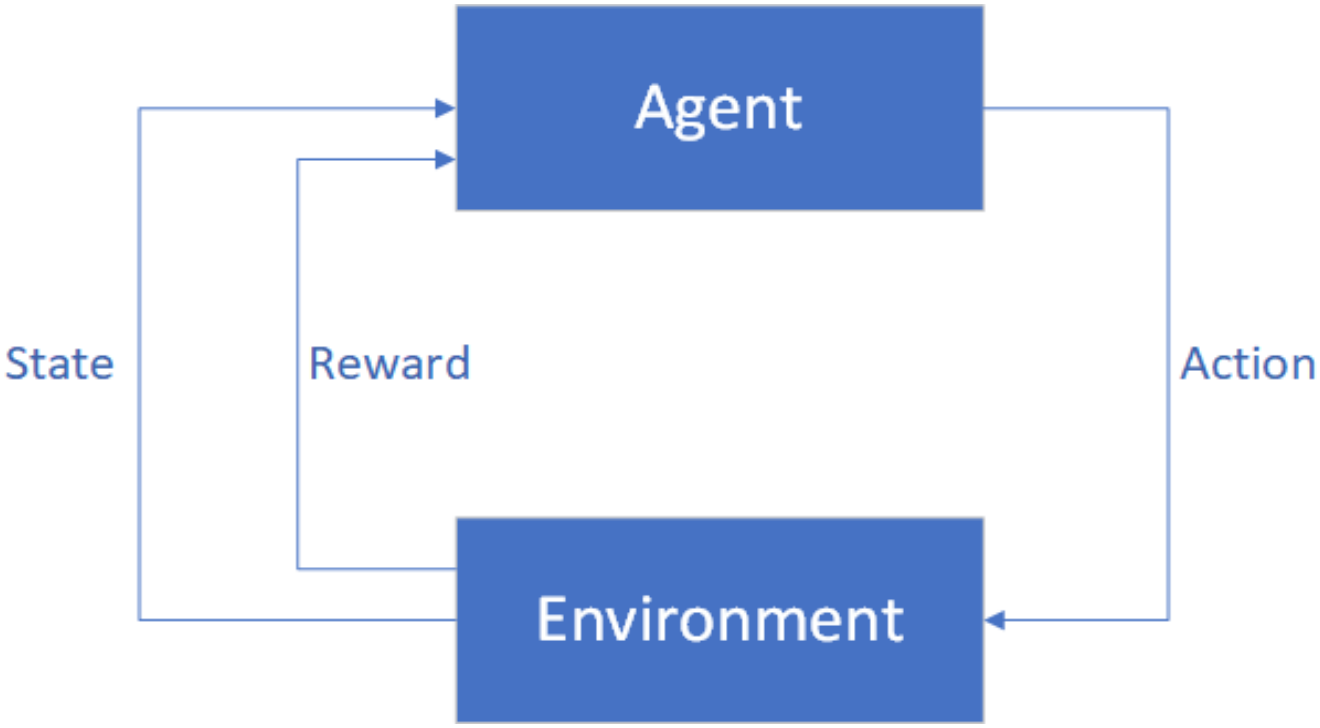
Step 2: Q-Learning Implementation

1. Initialize the Q-values for each state-action pair in the grid.

2. Define hyperparameters such as learning rate (alpha), discount factor (gamma), and exploration probability (epsilon).

Step 3: Implement the Q-Learning algorithm using a loop:

- Choose an action based on exploration/exploitation strategy (e.g., epsilon-greedy).
- Perform the chosen action and observe the new state and reward.
- Update the Q-value of the current state-action pair using the Q-Learning formula.
- Move to the new state and repeat the process.



Step 4: Exploration and Learning

1. Run multiple episodes of the Q-Learning algorithm to allow the agent to explore the environment.

2. Monitor how the Q-values are updated over episodes.

Step 5: Testing and Visualization

1. Once the Q-values have converged or after a sufficient number of episodes, test the learned policy by running the agent through the maze.

2. Visualize the agent's path and actions taken in the maze.

Step 6: Conclusion

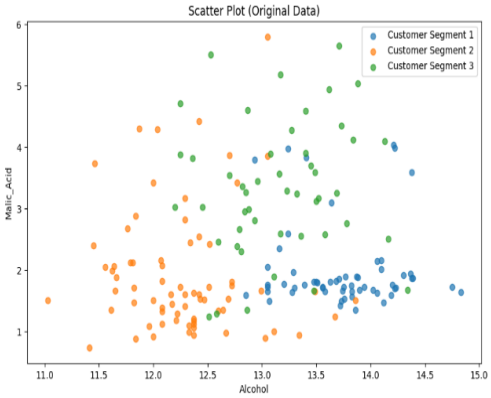
1. Summarize the results obtained from the Q-Learning algorithm.

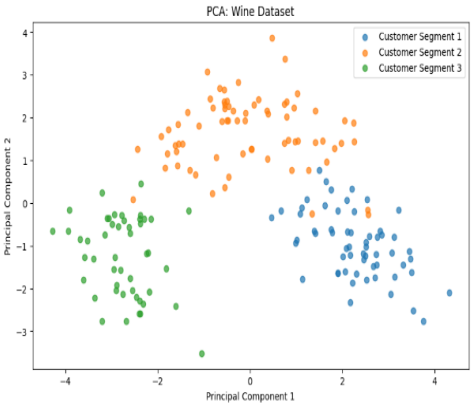
2. Discuss the effectiveness of the learned policy in navigating the maze.

3. Reflect on the exploration-exploitation trade-off and the role of hyperparameters.

Conclusion:

In this example, we successfully implemented Q-Learning for solving a maze navigation problem. By allowing the agent to learn an optimal policy through exploration and exploitation, demonstrated the principles of reinforcement learning. This example showcases how reinforcement learning can be used to train agents to make decisions in complex environments.

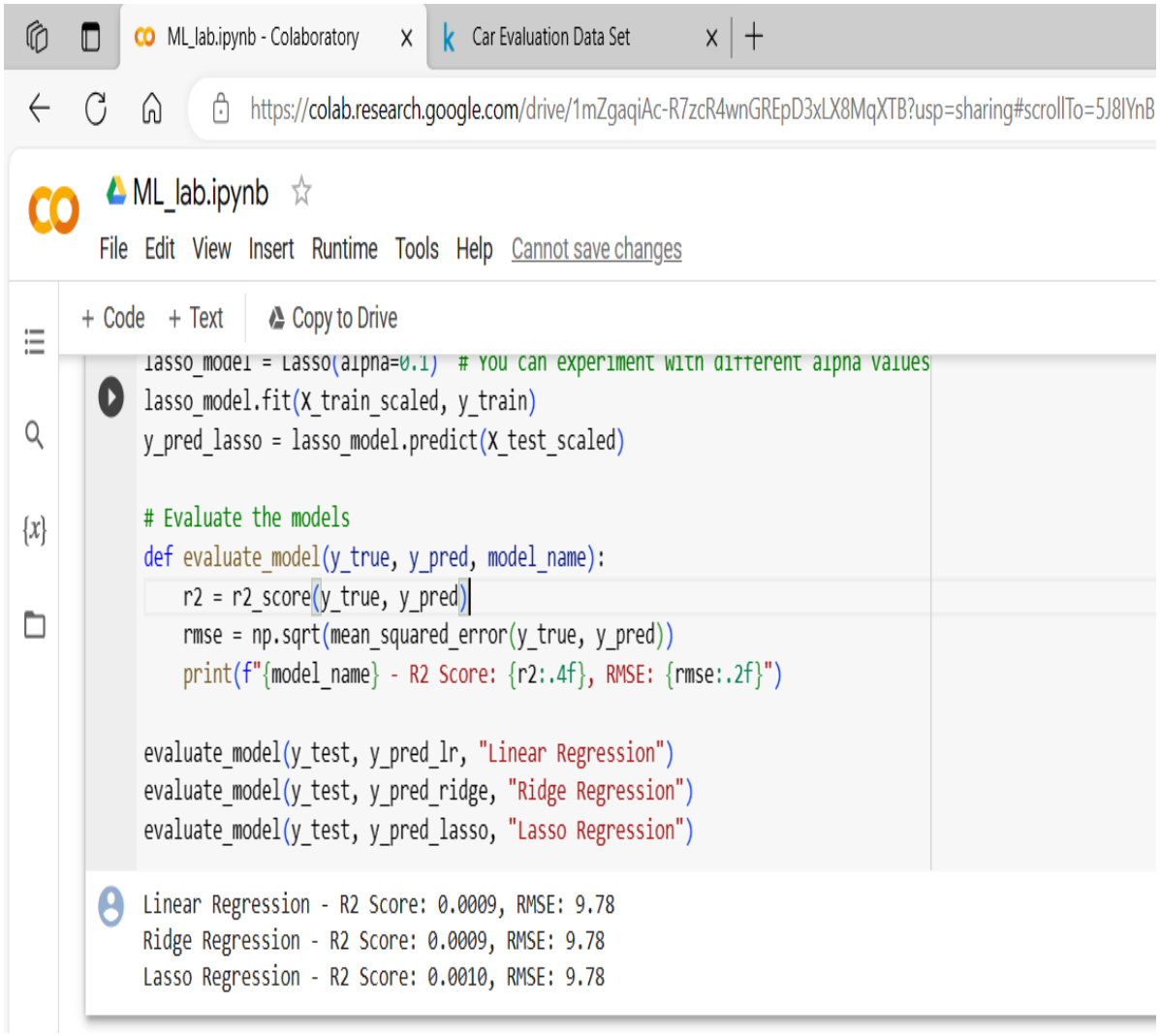





```
Accuracy: 1.0
Confusion Matrix:
[[14  0  0]
 [ 0 14  0]
 [ 0  0  8]]
Classification Report:
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        14
           2       1.00      1.00      1.00        14
           3       1.00      1.00      1.00         8

    accuracy                           1.00        36
   macro avg       1.00      1.00      1.00        36
weighted avg       1.00      1.00      1.00        36
```


ML_lab.ipynb - Colaboratory
Car Evaluation Data Set
https://colab.research.google.com/drive/1mZgaqiAc-R7zcR4wnGREpD3xLX8MqXTB?usp=sharing#scrollTo=5J8IYnB
ML_lab.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
+ Code + Text Copy to Drive
lasso_model = Lasso(alpha=0.1) # You can experiment with different alpha values
lasso_model.fit(X_train_scaled, y_train)
y_pred_lasso = lasso_model.predict(X_test_scaled)
# Evaluate the models
def evaluate_model(y_true, y_pred, model_name):
r2 = r2_score(y_true, y_pred)
rmse = np.sqrt(mean_squared_error(y_true, y_pred))
print(f"{model_name} - R2 Score: {r2:.4f}, RMSE: {rmse:.2f}")
evaluate_model(y_test, y_pred_lr, "Linear Regression")
evaluate_model(y_test, y_pred_ridge, "Ridge Regression")
evaluate_model(y_test, y_pred_lasso, "Lasso Regression")
Linear Regression - R2 Score: 0.0009, RMSE: 9.78
Ridge Regression - R2 Score: 0.0009, RMSE: 9.78
Lasso Regression - R2 Score: 0.0010, RMSE: 9.78

Pract 1ipynb - Colaboratory | Untitled6.ipynb - Colaboratory

colab.research.google.com/drive/1iuzd5hV4o7qB6i5OLyT1-aznF05U-Uf3

Untitled6.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

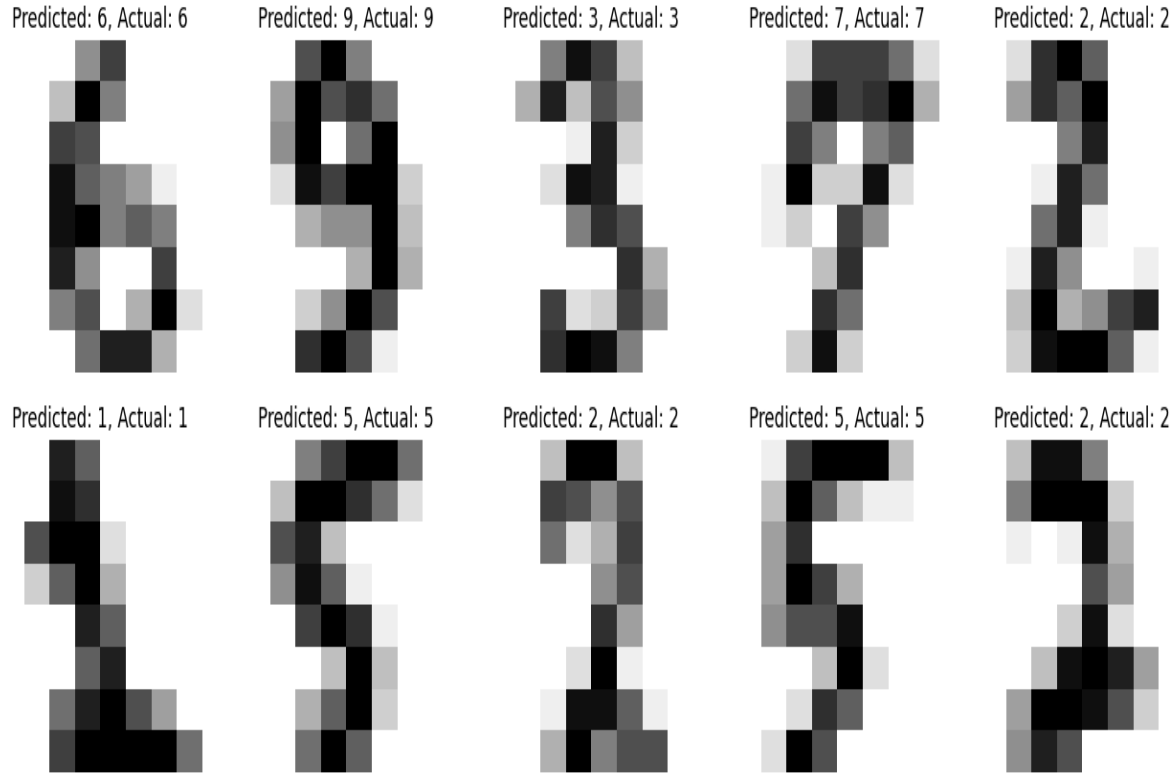
RAM Disk

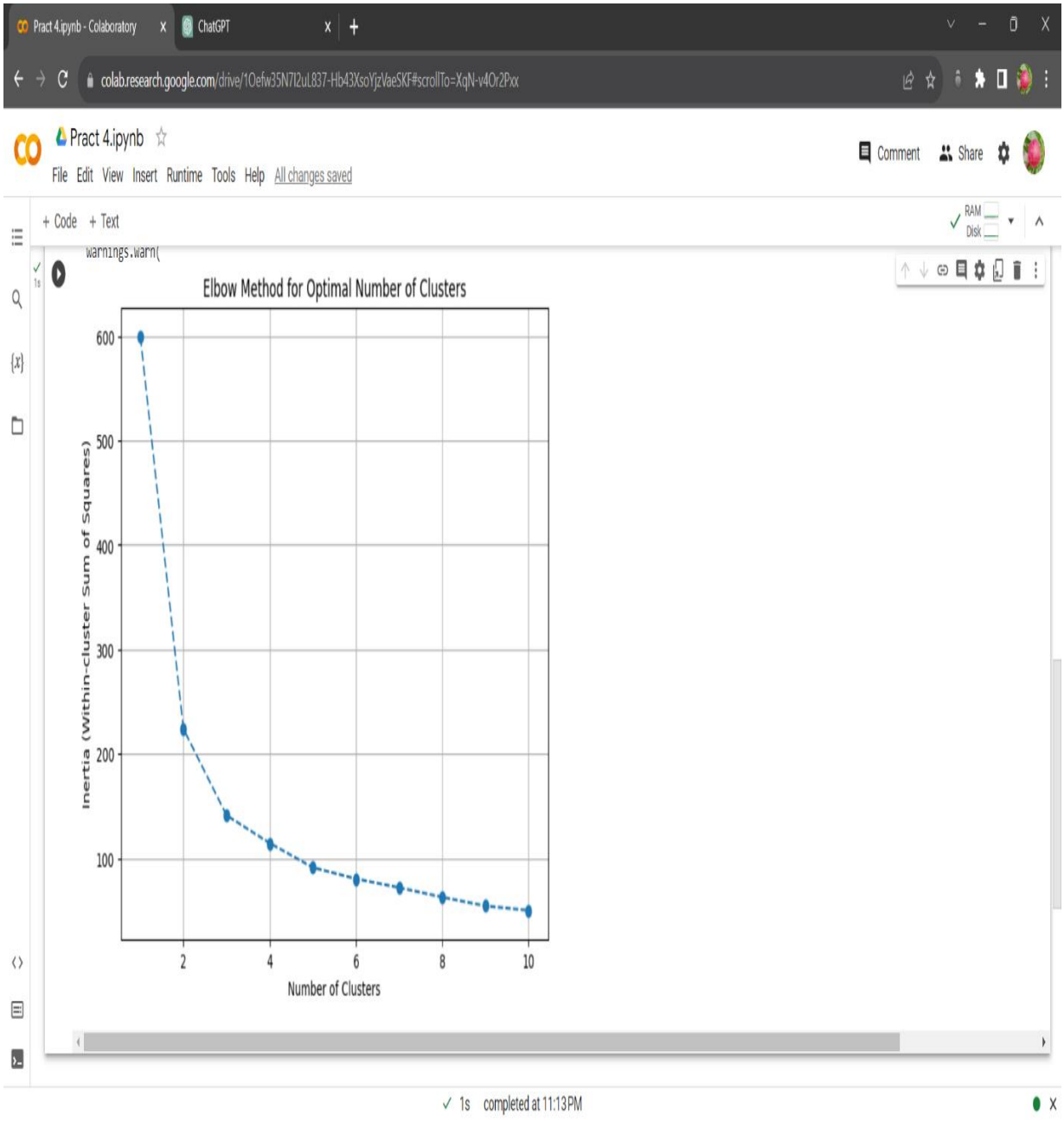
```
plt.show()
```

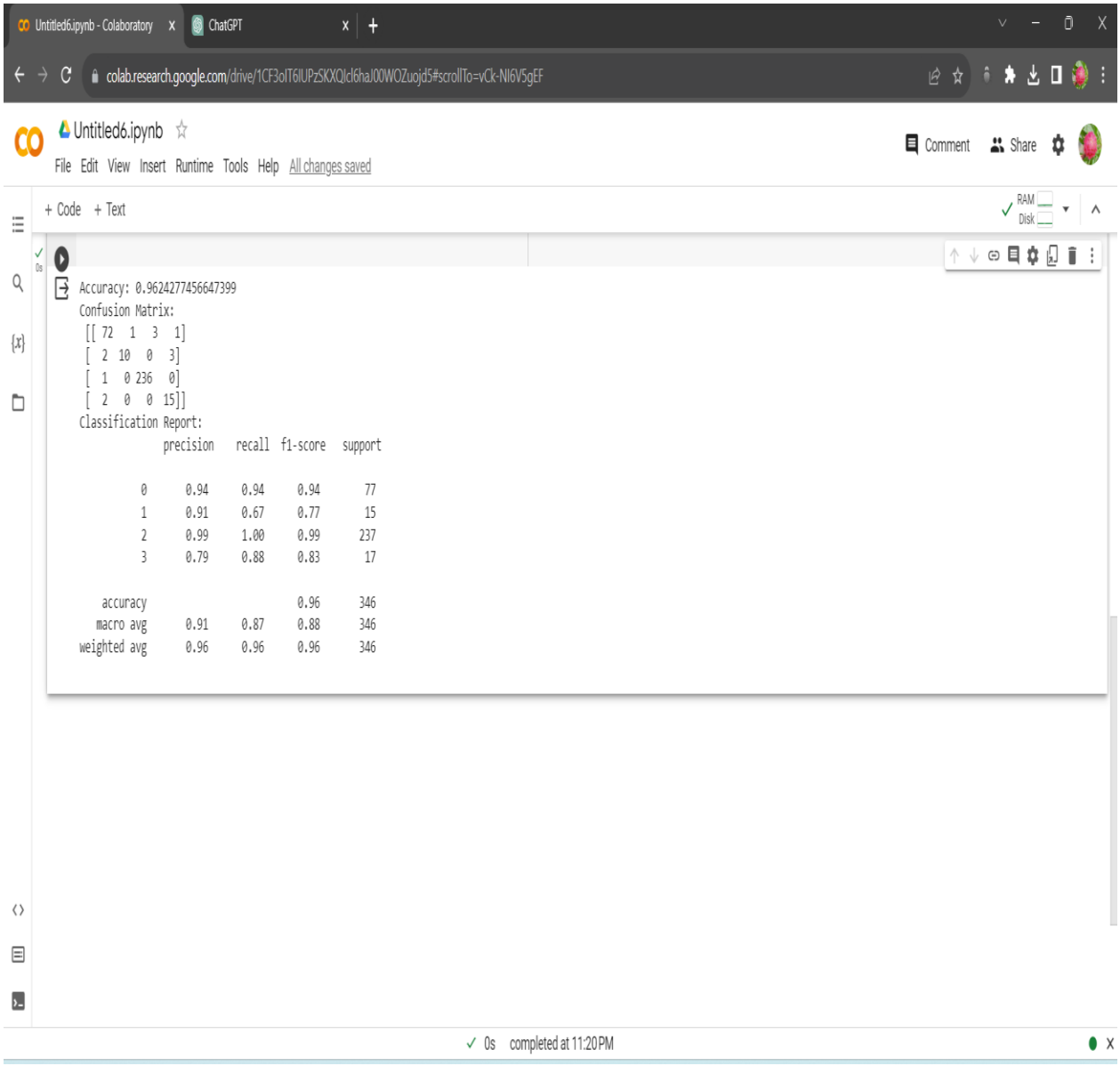
```
Accuracy: 0.9777777777777777
Confusion Matrix:
[[33  0  0  0  0  0  0  0  0  0]
 [ 0 28  0  0  0  0  0  0  0  0]
 [ 0  0 33  0  0  0  0  0  0  0]
 [ 0  0  0 32  0  1  0  0  0  1]
 [ 0  1  0  0 45  0  0  0  0  0]
 [ 0  0  0  0  0 47  0  0  0  0]
 [ 0  0  0  0  0  0 35  0  0  0]
 [ 0  0  0  0  0  0  0 33  0  1]
 [ 0  0  0  0  0  1  0  0 29  0]
 [ 0  0  0  1  1  0  0  1  0 37]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        33
           1       0.97      1.00      0.98        28
           2       1.00      1.00      1.00        33
           3       0.97      0.94      0.96        34
           4       0.98      0.98      0.98        46
           5       0.96      1.00      0.98        47
           6       1.00      1.00      1.00        35
           7       0.97      0.97      0.97        34
           8       1.00      0.97      0.98        30
           9       0.95      0.93      0.94        40

    accuracy                           0.98       360
   macro avg       0.98      0.98      0.98       360
weighted avg       0.98      0.98      0.98       360
```




Pract 4.ipynb - Colaboratory
ChatGPT
colab.research.google.com/drive/1Oefw35N7l2uL837-Hb43XsoYjzVaeSKF#scrollTo=XqN-v4Or2Pxx
Pract 4.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
warnings.warn(
Elbow Method for Optimal Number of Clusters
Inertia (Within-cluster Sum of Squares)
600
500
400
300
200
100
2
4
6
8
10
Number of Clusters
✓ 1s completed at 11:13 PM


Untitled6.ipynb - Colaboratory
ChatGPT
colab.research.google.com/drive/1CF3oIT6lUPzSKXQIcl6haJ00WOZuojd5#scrollTo=vCk-NI6V5gEF
Untitled6.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Accuracy: 0.9624277456647399
Confusion Matrix:
[[72 1 3 1]
[2 10 0 3]
[1 0 236 0]
[2 0 0 15]]
Classification Report:
precision recall f1-score support
0 0.94 0.94 0.94 77
1 0.91 0.67 0.77 15
2 0.99 1.00 0.99 237
3 0.79 0.88 0.83 17
accuracy 0.96 346
macro avg 0.91 0.87 0.88 346
weighted avg 0.96 0.96 0.96 346
0s completed at 11:20PM