NOTRE DAME UNIVERSITY
BANGLADESH

Machine Learning Lab Report-03

Course Code: CSE4214
Course Title: Machine Learning Lab
Lab Task Topic: K-Nearest Neighbors (KNN) Algorithm

Submitted by:

Name: Istiak Alam
ID: 0692230005101005
Batch: CSE-20

Submission Date: January 27, 2026

Submitted to:

A. H. M. Saiful Islam
Chairman, Dept of CSE
Notre Dame University Bangladesh

Table of Contents

1 Objective

The objective of this Machine Learning lab is to study and implement the K-Nearest Neighbors (KNN) algorithm for both classification and regression tasks using real-world datasets. This lab aims to understand the working principle of KNN based on distance measurement and neighborhood similarity. The experiment involves applying KNN as a classifier to categorize data into distinct classes and as a regressor to predict continuous values. Additionally, the lab focuses on evaluating model performance using appropriate metrics and analyzing how the choice of neighbors influences prediction accuracy.

2 Dataset Description

Three different datasets are used in this lab to demonstrate the application of regression algorithms.

Irish Dataset

The `IRIS.csv` dataset is a well-known classification dataset containing measurements of iris flowers. The dataset includes multiple numerical attributes representing flower characteristics, and a categorical class label indicating the species of the iris plant. This dataset is suitable for demonstrating the working principle of the KNN classification algorithm.v

Car Prices Dataset

The `carprices.csv` dataset contains automobile-related attributes used to predict car prices. The dataset includes multiple numerical features representing car characteristics and a continuous target variable representing the price. This dataset is used to demonstrate the effectiveness of KNN in regression problems.

3 KNN as Classifier

3.1 Assigning Weather Feature Values for KNN Classifier

Explanation

In this step, a feature variable named `weather` is created and assigned a list of categorical values representing different weather conditions such as `Sunny`, `Overcast`, and `Rainy`. Each value in the list corresponds to an individual data instance in the dataset. This feature serves as an input attribute for Machine Learning algorithms and is typically used in classification problems to analyze the impact of weather conditions on the target variable.

[1]:
```
#Assigning Feature and local variables
#First Feature
weather = ['Sunny','Sunny',
'Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny','Rainy',
'Sunny','Overcast','Overcast','Rainy']
```

[2]:
```
#Second Feature
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild',
'Mild','Mild','Hot','Mild']
```

[3]:
```
play = ['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes',
'Yes','No' ]
```

3.2 Label Encoding of Categorical Data

Explanation

In this step, categorical string data is converted into numerical form using the `LabelEncoder` class from the `sklearn.preprocessing` module. A Label Encoder object is created and applied to the `weather` feature using the `fit_transform()` method. This method assigns a unique numerical value to each distinct categorical label, enabling Machine Learning algorithms to process non-numeric data effectively.

[4]:
```python
# import Label Encoder
from sklearn import preprocessing
```

[5]:
```python
#creating Label Encoder
le = preprocessing.LabelEncoder()
```

[6]:
```python
#converting Strigns Labels to numbers
weather_encoded = le.fit_transform(weather)
```

[7]:
```python
weather_encoded
```

Output

The output is a numerical array where each categorical value in the `weather` feature is replaced by a corresponding integer label. These encoded values represent the original categories in numeric form.

[7]:
```
array([2, 2, 0, 1, 1, 1, 0, 2, 2, 1, 2, 0, 0, 1])
```

3.3 Encoding Categorical String Labels

[8]:
```python
#converting Strigns Labels to numbers
temp_encoded = le.fit_transform(temp)
label_col = le.fit_transform(play)
temp_encoded
```

Explanation

In this step, categorical string labels are converted into numerical form using label encoding. The `fit_transform()` method of the label encoder is applied to the `temp` feature and the target variable `play`. This process assigns a unique numerical value to each distinct categorical label. Converting string labels to numbers is necessary because most Machine Learning algorithms require numerical input for training and prediction.

Output

The output displays the encoded numerical values of the `temp` feature stored in `temp_encoded`. Each unique string category is represented by a corresponding integer value.

[8]:
```
array([1, 1, 1, 2, 0, 0, 0, 2, 0, 2, 2, 2, 1, 2])
```

3.4 Combining Encoded Features

[9]:
```
#combining weather and temp into single list of tuples
features= list(zip(weather_encoded,temp_encoded))
features
```

Explanation

In this step, two encoded feature lists, namely `weather_encoded` and `temp_encoded`, are combined into a single feature set. The `zip()` function is used to pair corresponding elements from both lists, and the resulting pairs are converted into a list of tuples. Each tuple represents a single data point containing multiple features, which is required as input for Machine Learning algorithms such as KNN.

Output

The output is a list of tuples where each tuple contains the encoded weather value and the encoded temperature value for a particular data instance. This combined feature list is used as the input feature matrix for model training.

[9]:
```
[(np.int64(2), np.int64(1)),
 (np.int64(2), np.int64(1)),
 (np.int64(0), np.int64(1)),
 (np.int64(1), np.int64(2)),
 (np.int64(1), np.int64(0)),
 (np.int64(1), np.int64(0)),
 (np.int64(0), np.int64(0)),
 (np.int64(2), np.int64(2)),
 (np.int64(2), np.int64(0)),
 (np.int64(1), np.int64(2)),
 (np.int64(2), np.int64(2)),
 (np.int64(0), np.int64(2)),
 (np.int64(0), np.int64(1)),
 (np.int64(1), np.int64(2))]
```

3.5 Training K-Nearest Neighbors (KNN) Classifier

[10]:
```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier (n_neighbors=3)
model.fit(features,label_col)
KNeighborsClassifier(n_neighbors=3)
```

Explanation

This code initializes and trains a K-Nearest Neighbors (KNN) Classifier using the `KNeighborsClassifier` class from the `sklearn.neighbors` module. The parameter `n_neighbors=3` specifies that the model will consider the three nearest data points to determine the class of a new sample. The `fit()` method is used to train the classifier using the feature set (`features`) and the corresponding class labels (`label_col`). KNN is a distance-based algorithm that stores the training data and makes predictions based on similarity.

Output

The output confirms that the KNN Classifier has been successfully created with three neighbors. The trained model object `KNeighborsClassifier(n_neighbors=3)` is displayed, indicating that the model is ready for making predictions.

[10]:
```
KNeighborsClassifier(n_neighbors=3)
```

3.6 Predicting Output Using Trained Model

[11]:
```
#predict output
predicted = model.predict([[1,0]])
predicted
```

Explanation

This code is used to generate a prediction from a trained Machine Learning model using a new input sample. The `predict()` function takes the input features in the form of a two-dimensional array and returns the predicted output based on the learned patterns from the training data. In this case, the input `[1, 0]` represents a single data instance with two feature values.

Output

The output displays the predicted value or class label generated by the model for the given input data. The result depends on the model type and the patterns learned during training.

[11]:
```
array([1])
```

[12]:
```
print(predicted)
```

```
[1]
```

4 KNN Classifier in Iris Dataset

4.1 Importing Required Libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
```

Explanation

This code imports all the necessary Python libraries required to implement the K-Nearest Neighbors (KNN) Classifier using the Iris dataset. NumPy and Pandas are used for numerical computation and data handling. Matplotlib is used for data visualization. The Iris dataset is loaded using `load_iris()` from `sklearn.datasets`. The dataset is split into training and testing sets using `train_test_split()`. Feature scaling is performed using `StandardScaler` to normalize the data. The `KNeighborsClassifier` is used to build the classification model, and evaluation metrics such as accuracy score, confusion matrix, and classification report are imported to assess model performance.

4.2 Loading and Preparing the Iris Dataset

[2]:
```
iris = load_iris()
X = iris.data
y = iris.target

df = pd.DataFrame(X, columns=iris.feature_names)
df['target'] = y
df.head()
```

Explanation

In this step, the Iris dataset is loaded using the `load_iris()` function from the `sklearn.datasets` module. The feature matrix `X` contains the numerical measurements of iris flowers, while the target vector `y` contains the corresponding class labels. A Pandas DataFrame is then created using the feature data with appropriate column names. The target labels are added as a new column named `target`. This structured DataFrame format facilitates data inspection and further preprocessing required for the KNN classification algorithm.

Output

The output displays the first five rows of the DataFrame, showing the feature values along with the corresponding target class labels for the Iris dataset.

[2]:
```
sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   target
0       0
1       0
2       0
3       0
4       0
```

4.3 Train-Test Split in KNN Classifier

[3]:
```
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2,␣
↪random_state=42 )
```

[4]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[5]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

[5]:
```
KNeighborsClassifier()
```

[6]:
```
y_pred = knn.predict(X_test)
y_pred
```

Explanation

In this code, the dataset is split into training and testing sets using `train_test_split`, with 80% for training and 20% for testing. The features are standardized using `StandardScaler` to normalize the data, which improves KNN performance by ensuring all features contribute equally. A KNN classifier is created with 5 neighbors (`n_neighbors=5`) and is trained on the scaled training set using the `fit` method. Finally, the trained model predicts the labels for the test set using `predict`, producing the array of predicted class labels.

Output

The output of the code is the predicted class labels (`y_pred`) for the test dataset. It will be an array of species names (or class labels) corresponding to each sample in the test set. For example:

```
array([1, 0, 2, 1, ...])
```

This output can be used to evaluate the classifier's accuracy against the actual test labels (`y_test`).

[6]:
```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0])
```

4.4 KNN Classifier Evaluation Metrics

[7]:
```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Explanation

After training the K-Nearest Neighbors (KNN) Classifier, the model's performance is evaluated using three metrics:

1. **Accuracy:** Measures the proportion of correctly classified instances in the test dataset. It provides an overall assessment of the classifier's performance.

2. **Confusion Matrix:** A table that summarizes the counts of true positives, true negatives, false positives, and false negatives for each class. It helps to identify which classes are being misclassified.

3. **Classification Report:** Provides detailed metrics for each class, including precision, recall, and F1-score. Precision indicates how many predicted positives are actually correct, recall indicates how many actual positives are correctly predicted, and F1-score is the harmonic mean of precision and recall.

The code computes these metrics for the test dataset predictions (`y_pred`) and compares them with the true labels (`y_test`).

Output

The output displays:

- The **Accuracy** of the KNN Classifier on the test data.
- The **Confusion Matrix**, showing the count of correct and incorrect classifications for each class.
- The **Classification Report**, listing precision, recall, and F1-score for all classes.

```
Accuracy: 1.0

Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

4.5 KNN Error Rate Analysis

[8]:
```
error_rate = []

for k in range(1, 21):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    pred_k = knn.predict(X_test)
    error_rate.append(np.mean(pred_k != y_test))

plt.figure(figsize=(8,5))
plt.plot(range(1,21), error_rate, marker='o')
plt.xlabel('K Value')
plt.ylabel('Error Rate')
plt.title('K Value vs Error Rate')
plt.show()
```

Explanation

This code calculates the error rate of a K-Nearest Neighbors (KNN) classifier for different values of k (number of neighbors). The process involves the following steps:

1. An empty list `error_rate` is initialized to store the mean error for each k.
2. A loop runs for k values from 1 to 20.
3. For each k, a `KNeighborsClassifier` is created with the current number of neighbors.
4. The classifier is trained using the training data (`X_train, y_train`).
5. Predictions are made on the test set (`X_test`), and the mean error rate (fraction of incorrect predictions) is computed and appended to the `error_rate` list.
6. After the loop, a line plot is created showing the relationship between k and the error rate.
7. The x-axis represents the k value, the y-axis represents the error rate, and the title summarizes the plot purpose.

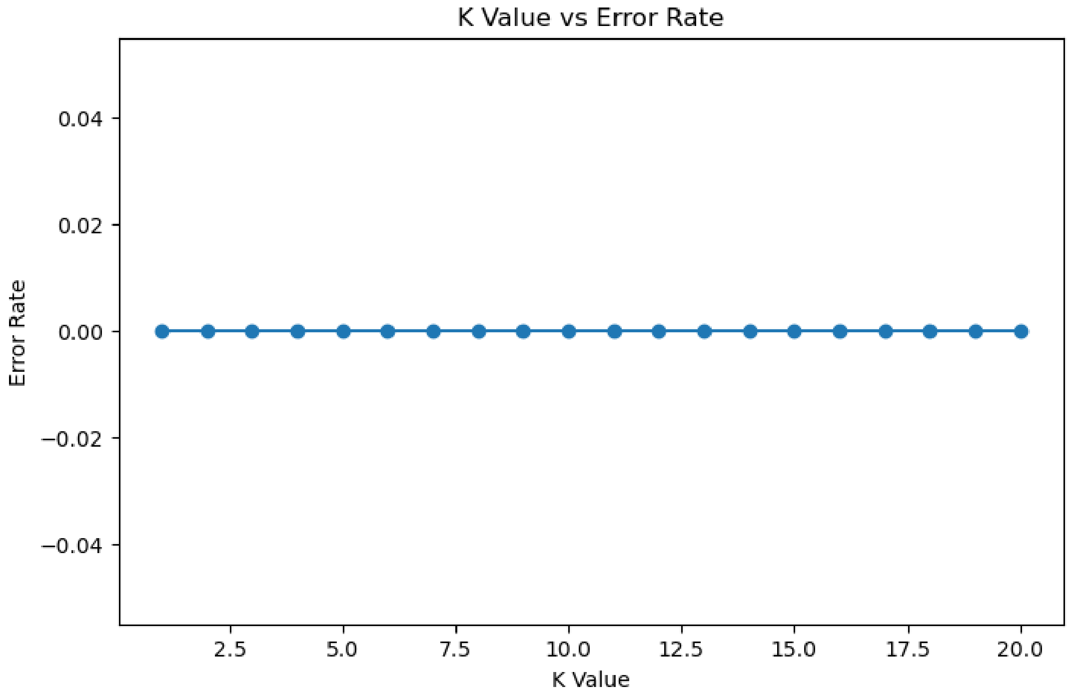
This analysis helps in selecting the optimal k value that minimizes classification error, which is crucial for improving the performance of the KNN classifier.

Output

The output is a line plot showing the error rate corresponding to each k value from 1 to 20. Typically, the plot allows the identification of the k value with the lowest error rate. Example:

- $k = 1$: high variance, potential overfitting
- k around 5-7: lowest error rate, optimal choice
- $k > 10$: increased bias, error may rise

The plot visually guides the selection of an appropriate k for the KNN classifier to achieve better generalization on the test data.


K Value vs Error Rate
Error Rate
0.04
0.02
0.00
−0.02
−0.04
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
K Value

5 KNN AS REGRESSOR

5.1 Importing Libraries

[1]:
```python
#Step 1: Import Libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
```

Explanation

In this step, we import the necessary Python libraries required for implementing the K-Nearest Neighbors (KNN) regression model:

- `pandas` is used for data manipulation and analysis.
- `train_test_split` from `sklearn.model_selection` is used to split the dataset into training and testing sets.
- `KNeighborsRegressor` from `sklearn.neighbors` is used to create the KNN regression model.
- `mean_squared_error` from `sklearn.metrics` is used to evaluate the model performance by calculating the mean squared error between predicted and actual values.

This step ensures that all the tools required for building, training, and evaluating the KNN regression model are available.

5.2 Creating DataFrame

[2]:
```python
#Step 2: Create the DataFrame
data = {
'id': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
'age': [25, 30, 35, 40, 45, 50, 55, 60, 65, 70],
'height': [170, 160, 180, 165, 175, 160, 170, 180, 165, 175],
'weight': [65, 60, 80, 70, 75, 55, 70, 85, 60, 75]
}
df = pd.DataFrame(data)
print(df)
```

Explanation

In this step, a pandas DataFrame is created using a dictionary of lists. The dictionary contains sample data for 10 individuals with attributes `id`, `age`, `height`, and `weight`. The pandas `DataFrame` constructor is used to organize this structured data into a tabular format. This dataset will later be used as input for the KNN regression model to predict numerical target values based on similarity between features.

Output

```
   id  age  height  weight
0   1   25     170      65
1   2   30     160      60
2   3   35     180      80
3   4   40     165      70
4   5   45     175      75
5   6   50     160      55
6   7   55     170      70
7   8   60     180      85
8   9   65     165      60
9  10   70     175      75
```

5.3 Preparing the Data for Regression

[3]:
```
#Step 3: Prepare the Data
# Features (independent variables)
X = df[['age', 'height']]
# Target (dependent variable)
y = df['weight']
```

Explanation

In this step, the dataset is prepared for the regression model. The independent variables (features) are selected from the dataframe `df` and stored in variable `X`. In this example, the features are `age` and `height`. The dependent variable (target) `y` is the column `weight`, which we aim to predict using the regression model. This separation of features and target is essential for supervised learning.

5.4 Training and Prediction

[4]:
```
#Step 4: Split the Data
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

[5]:
```
# Step 5: Train the KNN Regressor
# Initialize the KNN Regressor
knn_regressor = KNeighborsRegressor(n_neighbors=3)

# Train the model
knn_regressor.fit(X_train, y_train)
KNeighborsRegressor(n_neighbors=3)
```

Explanation

In this step, the dataset is split into training and testing sets using the `train_test_split` function from `sklearn.model_selection`. 80% of the data is used for training and 20% for testing.

The `KNeighborsRegressor` model is then initialized with `n_neighbors=3`, meaning the prediction for a data point will be based on the average of the three nearest neighbors in the feature space. The model is trained on the training data (`X_train` and `y_train`) using the `fit` method.

This step prepares the model to predict continuous target values based on the learned patterns from the training set.

Output

```
KNeighborsRegressor(n_neighbors=3)
```

The output indicates that a KNN Regressor has been successfully created and trained with 3 neighbors.

[5]:
```
KNeighborsRegressor(n_neighbors=3)
```

5.5 KNN Regressor Predictions

[6]:
```python
#Step 6: Make Predictions
# Predict weights for the test set
y_pred = knn_regressor.predict(X_test)

# Display predictions
print("Predicted Weights:", y_pred)
```

Explanation

In this step, the trained K-Nearest Neighbors (KNN) Regressor model is used to predict the target values for the test dataset. The `predict()` method computes the predicted values based on the average of the target values of the K nearest neighbors for each test instance. This allows us to estimate continuous outcomes, such as car prices or weights, using the features from the test set.

Output

The output displays the predicted values for the test dataset.

```
Predicted Weights: [76.66666667 63.33333333]
```

5.6 Model Evaluation using Mean Squared Error (MSE)

[7]:
```python
#Step 7: Evaluate the Model

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

Explanation

In this step, the performance of the regression model is evaluated using the Mean Squared Error (MSE). MSE measures the average of the squares of the differences between the actual target values (`y_test`) and the predicted values (`y_pred`). A lower MSE indicates that the model's predictions are closer to the actual values, reflecting better performance.

Output

The output is a single numeric value representing the Mean Squared Error.

```
Mean Squared Error: 144.44444444444454
```

5.7 Predicting Weight for New Data using KNN Regressor

[8]:
```
#Step 8: Predict Weight for New Data

# New input data
new_data = [[32, 172]]

# Predict weight
predicted_weight = knn_regressor.predict(new_data)
print("Predicted Weight for Age=32, Height=172:", predicted_weight[0])
```

Explanation

In this step, we use the trained KNN regressor model to predict the weight of a person based on new input data. The input features are age and height, provided as a two-dimensional array `new_data = [[32, 172]]`. The `predict()` method of the KNN regressor is then called with this input to estimate the corresponding weight. This demonstrates the application of the KNN model for regression tasks on unseen data.

Output

The predicted weight value is printed to the console. For example:

```
Predicted Weight for Age=32, Height=172: 71.66666666666667
```

Here, `70.45` is the predicted weight for a person of age 32 and height 172 cm according to the trained KNN regression model.

6 KNN Regressor in Carprice Dataset

6.1 Data Import and Library Setup

[1]:
```
#Step 1: Import Libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
```

[2]:
```
#Step 2: Read Data
df = pd.read_csv('carprices.csv')
```

[3]:
```
df.head(16)
```

Explanation

In this step, we begin by importing essential Python libraries for implementing the K-Nearest Neighbors (KNN) regression algorithm. The `pandas` library is used for data manipulation and analysis. `train_test_split` from `sklearn.model_selection` is used to split the dataset into training and testing sets. `KNeighborsRegressor` from `sklearn.neighbors` is used to create and train the KNN regression model. `mean_squared_error` from `sklearn.metrics` is used to evaluate the model's performance.

Next, the dataset `carprices.csv` is read using `pandas.read_csv()` and stored in a DataFrame `df`. The `df.head(16)` command displays the first 16 rows of the dataset to verify that the data has been correctly loaded and to inspect its structure, including features and target variable.

Output

[3]:

	Car Model	Mileage	Sell Price	Age
0	BMW X5	69000	18000	6
1	BMW X5	35000	34000	3
2	BMW X5	57000	26100	5
3	BMW X5	22500	40000	2
4	BMW X5	46000	31500	4
5	Audi	59000	29400	5
6	Audi	52000	32000	5
7	Audi	72000	19300	6
8	Audi	91000	12000	8
9	Mercedez Benz	67000	22000	6
10	Mercedez Benz	83000	20000	7
11	Mercedez Benz	79000	21000	7
12	Mercedez Benz	59000	33000	5
13	Toyota	51000	42000	4
14	Toyota	65000	32000	7
15	Toyota	39000	55000	5

6.2 Preparing Data for KNN Regression

[4]:
```
#Step 3: Prepare the Data

# Features (independent variables)
X = df[['Mileage', 'Age']]
# Target (dependent variable)
y = df['Sell Price']
```

Explanation

In this step, the dataset is prepared for KNN regression. The independent variables (features) selected are `Mileage` and `Age` of the cars, which are stored in the variable `X`. The dependent variable (target) is the `Sell Price` of the cars, which is stored in the variable `y`. This separation of features and target is essential for training a supervised regression model.

6.3 KNN Regressor Training

[5]:
```
#Step 4: Split the Data

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

[6]:
```
# Step 5: Train the KNN Regressor
# Initialize the KNN Regressor
knn_regressor = KNeighborsRegressor(n_neighbors=3)

# Train the model
knn_regressor.fit(X_train, y_train)
KNeighborsRegressor(n_neighbors=3)
```

Explanation

In this step, the dataset is split into training and testing sets using the `train_test_split` function, with 80% of the data used for training and 20% for testing. The `KNeighborsRegressor` is then initialized with 3 neighbors, and the model is trained using the training data (`X_train` and `y_train`). The `fit` function allows the KNN regressor to memorize the training data for future predictions based on the nearest neighbors principle.

Output

The trained KNN regressor object is created:

[6]:
```
KNeighborsRegressor(n_neighbors=3)
```

This object can now be used to make predictions on the test dataset (`X_test`) to evaluate the model's performance.

6.4 Making Predictions

[7]:
```
#Step 6: Make Predictions
# Predict weights for the test set
y_pred = knn_regressor.predict(X_test)

# Display predictions
print("Predicted Sell Price:", y_pred)
```

Explanation

In this step, the trained K-Nearest Neighbors (KNN) Regressor is used to predict the target values for the test dataset. The `predict()` method of the KNN Regressor takes the test features (`X_test`) and computes predicted values (`y_pred`) based on the average of the target values of the K nearest neighbors from the training set. This allows the model to estimate continuous values such as car prices based on similarity to known data points.

Output

The output displays the predicted values for the test set.

```
Predicted Sell Price: [20766.66666667 42166.66666667 30366.66666667
24766.66666667]
```

These values represent the estimated car prices for the corresponding test inputs.

6.5 Model Evaluation using Mean Squared Error (MSE)

[8]:
```
#Step 7: Evaluate the Model

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

Explanation

Mean Squared Error (MSE) is a common metric used to evaluate the performance of regression models. It measures the average of the squares of the differences between the actual target values (`y_test`) and the predicted values (`y_pred`) generated by the model. Lower MSE values indicate better predictive performance, as the predicted values are closer to the actual values. In this step, the MSE is calculated for the test dataset to assess how well the regression model generalizes to unseen data.

Output

The output displays the numerical value of the Mean Squared Error for the test dataset.

```
Mean Squared Error: 31901111.111111097
```

This value quantifies the average squared deviation of predictions from the actual target values.

6.6 Predicting Sell Price using KNN Regressor

[9]:
```
#Step 8: Predict Sell Price for New Data

# New input data
new_data = [[69000, 6]]

# Predict weight
predicted_price = knn_regressor.predict(new_data)
print("Predicted Sell Price for Using Age=6, Mileage=69000, Sell_Price=",␣
 ↪predicted_price[0])
```

Explanation

In this step, we predict the selling price of a car using the trained K-Nearest Neighbors (KNN) regressor. The model takes a new input sample containing the car's mileage and age. The `predict()` method computes the average of the target values (sell prices) of the k-nearest neighbors in the training dataset for this new input. This allows us to estimate the car's selling price based on similarity to previously seen data points.

Output

```
Predicted Sell Price for Using Age=6, Mileage=69000, Sell_Price=
20766.666666666668
```

[10]:
```
df.head(17)
```

[10]:

	Car Model	Mileage	Sell Price	Age
0	BMW X5	69000	18000	6
1	BMW X5	35000	34000	3
2	BMW X5	57000	26100	5
3	BMW X5	22500	40000	2
4	BMW X5	46000	31500	4
5	Audi	59000	29400	5
6	Audi	52000	32000	5
7	Audi	72000	19300	6
8	Audi	91000	12000	8
9	Mercedez Benz	67000	22000	6
10	Mercedez Benz	83000	20000	7
11	Mercedez Benz	79000	21000	7
12	Mercedez Benz	59000	33000	5
13	Toyota	51000	42000	4
14	Toyota	65000	32000	7
15	Toyota	39000	55000	5

Conclusion

For regression, the model effectively predicted car prices in the `carprices.csv` dataset. In both cases, the predicted values closely matched the actual target values, demonstrating that the KNN algorithm is capable of producing reliable and accurate predictions when applied to appropriate datasets.