krq2df0gt

September 13, 2023

[ ]:

```
# Q1. Write a Python code to implement the KNN classifier algorithm on␣
 ↪load_iris dataset in sklearn datasets.
```

[8]:

```
# Step 1: Import Necessary Libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Step 2: Load The Dataset
iris = load_iris()
X = iris.data
y = iris.target

# Step 3: Split The Dataset Into Training And Testing Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Step 4: Create And Train The KNN Classifier (Let's Use K=3 As An Example)
k = 3
knn_classifier = KNeighborsClassifier(n_neighbors=k)
knn_classifier.fit(X_train, y_train)

# Step 5: Make Predictions
y_pred = knn_classifier.predict(X_test)

# Step 6: Evaluate The Model's Performance (E.g., Using Accuracy)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of KNN classifier with K={k}: {accuracy:.2f}")
```

```
Accuracy of KNN classifier with K=3: 1.00
```

[ ]:

```
# Q2. Write a Python code to implement the KNN regressor algorithm on load␣
 ↪boston dataset in sklearn datasets.
```

[4]:

```
# load_boston has been removed from scikit-learn since version 1.2.
# load_boston has been removed from scikit-learn since version 1.2.
```

```python
# load_boston has been removed from scikit-learn since version 1.2.
# load_boston has been removed from scikit-learn since version 1.2.

# One such alternative is the California housing dataset
# One such alternative is the California housing dataset

import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Load The Boston Housing Dataset
california = fetch_california_housing()
X, y = california.data, california.target

# Split The Data Into Training And Testing Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Standardize The Feature Data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create A KNN Regressor With A Specified Number Of Neighbors (K)
k = 5
knn_regressor = KNeighborsRegressor(n_neighbors=k)

# Fit The Knn Regressor On The Training Data
knn_regressor.fit(X_train, y_train)

# Predict The Target Values On The Test Data
y_pred = knn_regressor.predict(X_test)

# Evaluate The Model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R-squared (R2) Score: {r2:.2f}")
```

```
Mean Squared Error (MSE): 0.43
R-squared (R2) Score: 0.67
```

[5]:

```
# Q3. Write a Python code snippet to find the optimal value of K for the KNN␣
↪classifier algorithm using cross-validation on load iris dataset in sklearn␣
↪datasets.
```

[10]:

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier


iris = load_iris()
X, y = iris.data, iris.target

# Create A KNN Classifier
knn_classifier = KNeighborsClassifier()

# Define A Range Of Values For K (Number Of Neighbors)
k_values = list(range(1, 31))  # You can adjust this range as needed

# Create A Parameter Grid For GridSearchCV
param_grid = {'n_neighbors': k_values}



# Perform A Grid Search With 5-fold Cross-validation To Find The Optimal K
grid_search = GridSearchCV(knn_classifier, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X, y)


# Get The Best K Value From The Grid Search
best_k = grid_search.best_params_['n_neighbors']
best_accuracy = grid_search.best_score_

print(f"The best K value is {best_k} with an accuracy of {best_accuracy:.2f}")
```

```
The best K value is 6 with an accuracy of 0.98
```

[ ]:

```
# Q4. Implement the KNN regressor algorithm with feature scaling on load boston␣
↪dataset in sklearn datasets.
```

[11]:

```
# load_boston has been removed from scikit-learn since version 1.2.
# load_boston has been removed from scikit-learn since version 1.2.
# load_boston has been removed from scikit-learn since version 1.2.
# load_boston has been removed from scikit-learn since version 1.2.

# One such alternative is the California housing dataset
# One such alternative is the California housing dataset
```

```python
import numpy as np
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Load The California Housing Dataset
california = fetch_california_housing()
X, y = california.data, california.target

# Split The Data Into Training And Testing Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Standardize The Feature Data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create A KNN Regressor With A Specified Number Of Neighbors (K)
k = 5
knn_regressor = KNeighborsRegressor(n_neighbors=k)

# Fit The Knn Regressor On The Training Data
knn_regressor.fit(X_train, y_train)

# Predict The Target Values On The Test Data
y_pred = knn_regressor.predict(X_test)

# Evaluate The Model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R-squared (R2) Score: {r2:.2f}")
```

```
Mean Squared Error (MSE): 0.43
R-squared (R2) Score: 0.67
```

[12]:

```python
# Q5. Write a Python code snippet to implement the KNN classifier algorithm␣
 ↪with weighted voting on load_iris dataset in sklearn.datasets.
```

[13]:

```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

# Load The Iris Dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split The Data Into Training And Testing Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Standardize The Feature Data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create A KNN Classifier With Weighted Voting
k = 5
knn_classifier = KNeighborsClassifier(n_neighbors=k, weights='distance')


# We Create A Knn Classifier With Weighted Voting By Specifying␣
 ↪Weights='distance'.
# ========> This Means That Closer Neighbors Have More Influence On The␣
 ↪Prediction.


# Fit The Knn Classifier On The Training Data
knn_classifier.fit(X_train, y_train)

# Predict The Class Labels On The Test Data
y_pred = knn_classifier.predict(X_test)

# Evaluate The Model
accuracy = np.mean(y_pred == y_test)

print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 1.00
```

[ ]:

```
# Q6. Implement a function to standardise the features before applying KNN␣
 ↪classifier.
```

[25]:

```
import numpy as np
import pandas as pd
```

```
def standardize_features(X):

    # Check If The Input Is A Pandas Dataframe And Convert It To A Numpy Array
    if isinstance(X, pd.DataFrame):
        X = X.values

    # Calculate The Mean And Standard Deviation For Each Feature
    mean = np.mean(X, axis=0)
    std_dev = np.std(X, axis=0)

    # Avoid Division By Zero By Setting Std_Dev To 1 For Features With Zero␣
 ↪Standard Deviation
    std_dev[std_dev == 0] = 1

    # Standardize The Features
    X_standardized = (X - mean) / std_dev

    return X_standardized
```

[26]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Load The Iris Dataset As An Example
iris = load_iris()
X = iris.data
y = iris.target

# Split The Data Into Training And Testing Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Standardize The Features
X_train_std = standardize_features(X_train)
X_test_std = standardize_features(X_test)

# Initialize And Fit The KNN Classifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train_std, y_train)

# Make Predictions On The Standardized Test Data
y_pred = knn.predict(X_test_std)

# Evaluate The Models Performance
accuracy = np.mean(y_pred == y_test)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9666666666666667
```

[ ]:
```
# Q7. Write a Python function to calculate the euclidean distance between two␣
  ↪points.
```

[21]:
```
import math

def euclidean_distance(x1, y1, x2, y2):
    return math.sqrt((x1 - x2)**2 + (y1 - y2)**2)

# Example usage:
point1 = (5, 5)
point2 = (1, 10)
distance = euclidean_distance(point1[0], point1[1], point2[0], point2[1])
print("Euclidean distance:", distance)
```

```
Euclidean distance: 6.4031242374328485
```

[ ]:
```
# Q8. Write a Python function to calculate the manhattan distance between two␣
  ↪points.
```

[22]:
```
def manhattan_distance(x1, y1, x2, y2):
    return abs(x1 - x2) + abs(y1 - y2)

# Example usage:
point1 = (5, 5)
point2 = (1, 10)
distance = manhattan_distance(point1[0], point1[1], point2[0], point2[1])
print("Manhattan distance:", distance)
```

```
Manhattan distance: 9
```