bmi-1

June 27, 2024

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import sklearn.model_selection as train_test_split
import sklearn.neighbors as KNeighborsClassifier
import sklearn.metrics as accuracy_score
```

[ ]:
```
data_bmi=pd.read_csv("/content/drive/MyDrive/bmi_train.csv")
```

[ ]:
```
x=data_bmi[['Height','Weight']]
y=data_bmi['Index']
```

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split  # Import the class␣
 ↪directly
from sklearn.neighbors import KNeighborsClassifier  # Import the class directly
from sklearn.metrics import accuracy_score

# ... (rest of your code)

k = 3
knn = KNeighborsClassifier(n_neighbors=k)  # Now you're using the class␣
 ↪correctly
knn.fit(x, y)
```

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

[ ]:
```
new_data = np.array([[173,82]])
prediction = knn.predict(new_data)




if prediction==0:
    print("Extremely Weak")
elif prediction==1:
```

```python
    print("Weak")
elif prediction==2:
    print("Normal")
elif prediction==3:
    print("Over Weight")
elif prediction==4:
    print("obesity")
else :
    print("extremely high obesity")
```

```
Normal
```

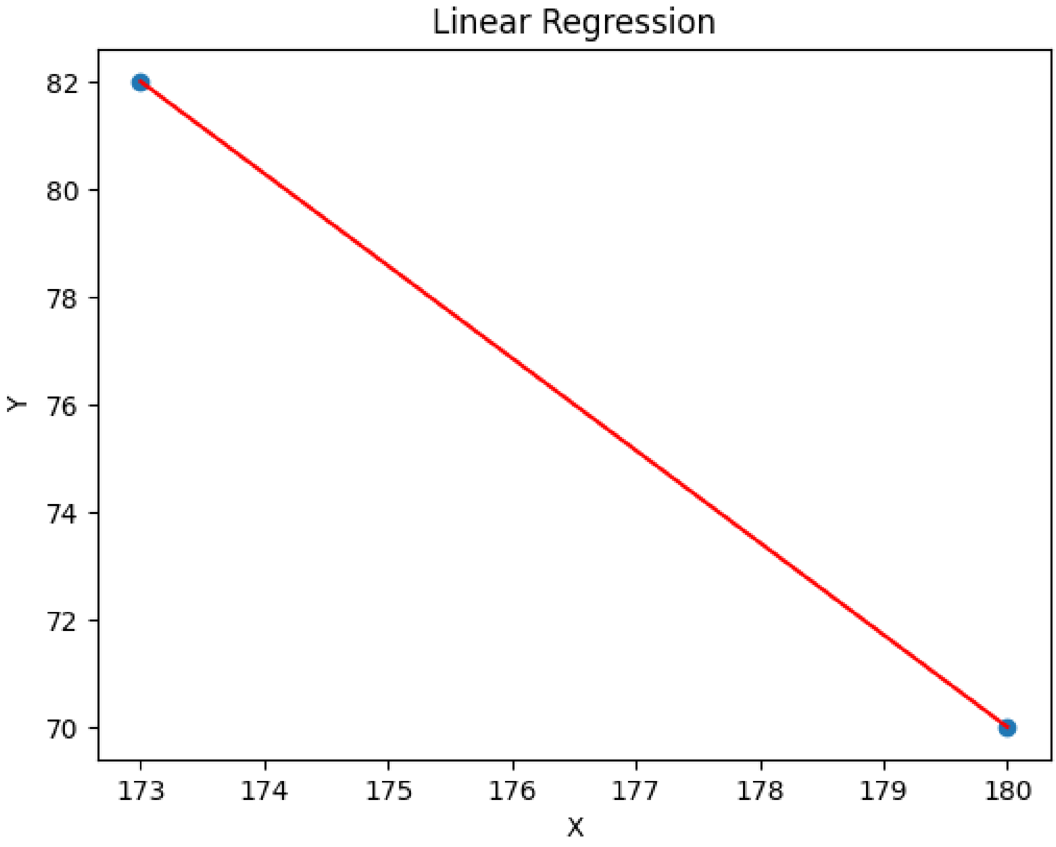
```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does
not have valid feature names, but KNeighborsClassifier was fitted with feature
names
  warnings.warn(
```

[ ]:

```python
# prompt: linear regression with user input

import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Get user input for data points
n = int(input("Enter the number of data points: "))
x = []
y = []
for i in range(n):
  xi = float(input(f"Enter x-coordinate for point {i+1}: "))
  yi = float(input(f"Enter y-coordinate for point {i+1}: "))
  x.append(xi)
  y.append(yi)

# Convert lists to NumPy arrays and reshape for sklearn
x = np.array(x).reshape(-1, 1)
y = np.array(y)

# Create and fit the linear regression model
model = LinearRegression()
model.fit(x, y)

# Get user input for prediction
new_x = float(input("Enter a new x-value to predict its y-value: "))
new_x = np.array([[new_x]])

# Make prediction
prediction = model.predict(new_x)
```

```
# Print results
print("Intercept:", model.intercept_)
print("Slope:", model.coef_[0])
print("Predicted y-value:", prediction[0])

# Plot the data and the regression line (optional)
plt.scatter(x, y)
plt.plot(x, model.predict(x), color='red')
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Linear Regression")
plt.show()
```

```
Enter the number of data points: 2
Enter x-coordinate for point 1: 173
Enter y-coordinate for point 1: 82
Enter x-coordinate for point 2: 180
Enter y-coordinate for point 2: 70
Enter a new x-value to predict its y-value: 173
Intercept: 378.5714285714286
Slope: -1.7142857142857146
Predicted y-value: 82.0
```



[ ]:

```python
# prompt: logistic regression with user input

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Get user input for data
n = int(input("Enter the number of data points (must be greater than 1): "))
# Ensure the user enters a number greater than 1
while n <= 1:
    print("Number of data points must be greater than 1.")
    n = int(input("Enter the number of data points (must be greater than 1): "))

features = []
labels = []
for i in range(n):
    feature_values = input(f"Enter feature values for data point {i+1}␣
 ↪(comma-separated): ").split(',')
    features.append([float(val) for val in feature_values])
    # Ensure the user enters either 0 or 1 for the label
    while True:
        label = int(input(f"Enter label (0 or 1) for data point {i+1}: "))
        if label in [0, 1]:
            break
        else:
            print("Invalid label. Please enter 0 or 1.")
    labels.append(label)

# Convert lists to NumPy arrays
X = np.array(features)
y = np.array(labels)

# Split data into training and testing sets, adjusting test_size if needed
# If you still have a small dataset, consider a smaller test_size or using␣
 ↪cross-validation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=0)

# Check if both classes are present in the training set
if len(np.unique(y_train)) < 2:
    print("Error: Both classes (0 and 1) are not present in the training set.␣
 ↪Please provide a more diverse dataset.")
```

```
else:
    # Create and fit the logistic regression model
    model = LogisticRegression()
    model.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = model.predict(X_test)

    # Evaluate the model
    accuracy = accuracy_score(y_test, y_pred)
    print("Accuracy:", accuracy)

    # Get user input for new data point
    new_data_point = input("Enter feature values for a new data point␣
 ↪(comma-separated): ").split(',')
    new_data_point = np.array([float(val) for val in new_data_point]).
 ↪reshape(1, -1)

    # Make prediction for the new data point
    prediction = model.predict(new_data_point)
    print("Prediction for new data point:", prediction[0])
```

```
Enter the number of data points (must be greater than 1): 3
Enter feature values for data point 1 (comma-separated): 194,108,3
Enter label (0 or 1) for data point 1: 1
Enter feature values for data point 2 (comma-separated): 142,159,5
Enter label (0 or 1) for data point 2: 0
Enter feature values for data point 3 (comma-separated): 151,64,3
Enter label (0 or 1) for data point 3: 1
Accuracy: 1.0
Enter feature values for a new data point (comma-separated): 185,102,3
Prediction for new data point: 1
```

[ ]:

```
# prompt: decision tree with user input

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np

# Get user input for data
n = int(input("Enter the number of data points: "))
features = []
labels = []
for i in range(n):
    feature_values = input(f"Enter feature values for data point {i+1}␣
 ↪(comma-separated): ").split(',')
```

```
    features.append([float(val) for val in feature_values])
    label = int(input(f"Enter label for data point {i+1}: "))
    labels.append(label)

# Convert lists to NumPy arrays
X = np.array(features)
y = np.array(labels)

# Split 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)

# Create and fit the decision tree model
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Get user input for new data point
new_data_point = input("Enter feature values for a new data point␣
 ↪(comma-separated): ").split(',')
new_data_point = np.array([float(val) for val in new_data_point]).reshape(1, -1)

# Make prediction for the new data point
prediction = model.predict(new_data_point)
print("Prediction for new data point:", prediction[0])
```

```
Enter the number of data points: 3
Enter feature values for data point 1 (comma-separated): 161,89,4
Enter label for data point 1: 1
Enter feature values for data point 2 (comma-separated): 179,127,4
Enter label for data point 2: 0
Enter feature values for data point 3 (comma-separated): 172,139,5
Enter label for data point 3: 1
Accuracy: 0.0
Enter feature values for a new data point (comma-separated): 142,159,5
Prediction for new data point: 1
```

[ ]:

```
# prompt: random forest with user input

import numpy as np
from sklearn.ensemble import RandomForestClassifier
```

```python
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Get user input for data
n = int(input("Enter the number of data points: "))
features = []
labels = []
for i in range(n):
    feature_values = input(f"Enter feature values for data point {i+1} (comma-separated): ").split(',')
    features.append([float(val) for val in feature_values])
    label = int(input(f"Enter label for data point {i+1}: "))
    labels.append(label)

# Convert lists to NumPy arrays
X = np.array(features)
y = np.array(labels)

# Split 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)

# Create and fit the Random Forest model
model = RandomForestClassifier(n_estimators=100)  # You can adjust the number of trees (n_estimators)
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Get user input for new data point
new_data_point = input("Enter feature values for a new data point (comma-separated): ").split(',')
new_data_point = np.array([float(val) for val in new_data_point]).reshape(1, -1)

# Make prediction for the new data point
prediction = model.predict(new_data_point)
print("Prediction for new data point:", prediction[0])
```

```
Enter the number of data points: 3
Enter feature values for data point 1 (comma-separated): 179,127,4
Enter label for data point 1: 0
Enter feature values for data point 2 (comma-separated): 155,57,2
Enter label for data point 2: 1
Enter feature values for data point 3 (comma-separated): 173,82,2
Enter label for data point 3: 1
Accuracy: 0.0
Enter feature values for a new data point (comma-separated): 157,56,2
Prediction for new data point: 1
```