22nd April Assignment

April 29, 2023

1 Assignment 75

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

[1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.datasets import load_iris
```

[2]:

```
dataset = load_iris()
```

[3]:

```
df = pd.DataFrame(data=dataset.data,columns=dataset.feature_names)
df['Target'] = dataset.target
df.head()
```

[3]:

```
   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]:

```
df.isnull().sum()
```

[4]:

```
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
```

```
Target           0
dtype: int64
```

[5]:

```
# segregate the data into independent and dependent features
X = df.iloc[:,:-1]
y = df['Target']
```

[6]:

```
# train test and split
from sklearn.model_selection import train_test_split
```

[8]:

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.
 ↪20,random_state=42)
```

[9]:

```
# model training
from sklearn.neighbors import KNeighborsClassifier
```

[10]:

```
classifier = KNeighborsClassifier(n_neighbors=5,algorithm='auto')
```

[11]:

```
classifier.fit(X_train,y_train)
```

[11]:

```
KNeighborsClassifier()
```

[12]:

```
y_pred = classifier.predict(X_test)
```

[13]:

```
# Evaluation of performance
from sklearn.metrics import confusion_matrix, accuracy_score,␣
 ↪classification_report
```

[14]:

```
print(confusion_matrix(y_pred,y_test))
print(accuracy_score(y_pred,y_test))
print(classification_report(y_pred,y_test))
```

```
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
1.0
              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
```

Q2. Write a Python code to implement the KNN regressor algorithm on load_boston dataset in sklearn.datasets.

```
[15]: import pandas as pd
      import numpy as np
      import seaborn as sns
      import matplotlib.pyplot as plt
      %matplotlib inline
      import urllib.request
```

```
[16]: # Download the dataset from its URL
      url = 'https://raw.githubusercontent.com/selva86/datasets/master/BostonHousing.
        ↪csv'
      urllib.request.urlretrieve(url, 'BostonHousing.csv')

      # Load the dataset into a Pandas DataFrame
      df = pd.read_csv('BostonHousing.csv')
```

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

```
[17]:       crim    zn  indus  chas    nox     rm   age     dis  rad  tax  ptratio  \
      0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3
      1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
      2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8
      3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7
      4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7

              b  lstat  medv
      0  396.90   4.98  24.0
      1  396.90   9.14  21.6
      2  392.83   4.03  34.7
      3  394.63   2.94  33.4
      4  396.90   5.33  36.2
```

```
[18]: df.isnull().sum()
```

```
[18]: crim       0
      zn         0
      indus      0
      chas       0
      nox        0
      rm         0
      age        0
      dis        0
      rad        0
      tax        0
      ptratio    0
      b          0
```

```
lstat      0
medv       0
dtype: int64
```

[19]:
```
# segregate the feature into independent and dependent feature
X = df.iloc[:,:-1]
y = df['medv']
```

[20]:
```
# train test and split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.
 ↪20,random_state=42)
```

[21]:
```
# model training
from sklearn.neighbors import KNeighborsRegressor
```

[22]:
```
reg = KNeighborsRegressor()
```

[23]:
```
reg.fit(X_train,y_train)
```

[23]:
```
KNeighborsRegressor()
```

[24]:
```
y_pred = reg.predict(X_test)
```

[25]:
```
# Evaluation of performance
from sklearn.metrics import r2_score
```

[26]:
```
print(r2_score(y_test,y_pred))
```

```
0.6473640882039258
```

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.

[27]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.datasets import load_iris
```

[28]:
```
dataset = load_iris()
```

[29]:
```
df = pd.DataFrame(data=dataset.data,columns=dataset.feature_names)
df['Target'] = dataset.target
df.head()
```

[29]:

```
   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
```

[30]:

```
# segregate the data into independent and dependent features
X = df.iloc[:,:-1]
y = df['Target']
```

[31]:

```
# train test and split
from sklearn.model_selection import train_test_split
```

[32]:

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.
 ↪20,random_state=42)
```

[33]:

```
# model training
from sklearn.neighbors import KNeighborsClassifier
```

[34]:

```
classifier = KNeighborsClassifier()
```

[35]:

```
# Task
from sklearn.model_selection import GridSearchCV
K_val = range(1,11)
parameters = {
    'n_neighbors':K_val
}
```

[36]:

```
# Hypertunning
from sklearn.model_selection import GridSearchCV
```

[37]:

```
grid = GridSearchCV(classifier,param_grid=parameters,scoring='accuracy')
```

[38]:

```
grid.fit(X_train,y_train)
```

[38]:

```
GridSearchCV(estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': range(1, 11)}, scoring='accuracy')
```

[40]:

```
grid.best_params_
```

```
[40]: {'n_neighbors': 3}
```

```
[41]: classifier= KNeighborsClassifier(**grid.best_params_)
```

```
[42]: classifier.fit(X_train,y_train)
```

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

```
[43]: y_pred = classifier.predict(X_test)
```

```
[44]: # Evaluation of performance
      from sklearn.metrics import confusion_matrix, accuracy_score,␣
       ↪classification_report
      print(confusion_matrix(y_pred,y_test))
      print(accuracy_score(y_pred,y_test))
      print(classification_report(y_pred,y_test))
```

```
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
1.0
              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
```

Q4. Implement the KNN regressor algorithm with feature scaling on load_boston dataset in sklearn.datasets.

```
[45]: import pandas as pd
      import numpy as np
      import seaborn as sns
      import matplotlib.pyplot as plt
      %matplotlib inline
      import urllib.request
```

```
[46]: # Download the dataset from its URL
      url = 'https://raw.githubusercontent.com/selva86/datasets/master/BostonHousing.
       ↪csv'
      urllib.request.urlretrieve(url, 'BostonHousing.csv')

      # Load the dataset into a Pandas DataFrame
```

```
df = pd.read_csv('BostonHousing.csv')
```

[47]:

```
df.head()
```

[47]:

```
      crim    zn  indus  chas    nox     rm   age     dis  rad  tax  ptratio  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15.3
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17.8
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17.8
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18.7
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18.7

        b  lstat  medv
0  396.90   4.98  24.0
1  396.90   9.14  21.6
2  392.83   4.03  34.7
3  394.63   2.94  33.4
4  396.90   5.33  36.2
```

[48]:

```
# segregate the feature into independent and dependent feature
X = df.iloc[:,:-1]
y = df['medv']
```

[49]:

```
# train test and split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.
 ↪20,random_state=42)
```

[50]:

```
# feature scaling
from sklearn.preprocessing import StandardScaler
```

[51]:

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[52]:

```
# model training
from sklearn.neighbors import KNeighborsRegressor
```

[53]:

```
# Train the KNN regressor
knn_regressor = KNeighborsRegressor(n_neighbors=5)
knn_regressor.fit(X_train_scaled, y_train)
```

[53]:

```
KNeighborsRegressor()
```

[54]:

```
# Make predictions on the test set
y_pred = knn_regressor.predict(X_test_scaled)
```

[55]:

```
# Evaluation of performance
from sklearn.metrics import mean_squared_error, r2_score
print(mean_squared_error(y_test,y_pred))
print(r2_score(y_test,y_pred))
```

```
20.60552941176471
0.7190172315709293
```

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

[56]:

```
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

# Load the Iris dataset
X, y = load_iris(return_X_y=True)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Train the KNN classifier with weighted voting
knn_classifier = KNeighborsClassifier(n_neighbors=5, weights='distance')
knn_classifier.fit(X_train, y_train)

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

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

```
Accuracy: 1.0
```

Q6. Implement a function to standardise the features before applying KNN classifier.

[57]:

```
from sklearn.preprocessing import StandardScaler

def knn_classifier(X_train, X_test, y_train, y_test, n_neighbors):
    # Standardize the features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Train the KNN classifier
    knn = KNeighborsClassifier(n_neighbors=n_neighbors)
    knn.fit(X_train_scaled, y_train)

    # Make predictions on the test set
    y_pred = knn.predict(X_test_scaled)

    # Return the predictions and the trained model
    return y_pred, knn
```

Q7. Write a Python function to calculate the euclidean distance between two points.

[58]:

```
import numpy as np
def euclidean_distance(point1, point2):
    point1 = np.asarray(point1)
    point2 = np.asarray(point2)
    distance = np.sqrt(np.sum((point1 - point2) ** 2))
    return distance
```

[59]:

```
point1 = [1, 2, 3]
point2 = [4, 5, 6]
distance = euclidean_distance(point1, point2)
print(distance)
```

```
5.196152422706632
```

Q8. Write a Python function to calculate the manhattan distance between two points.

[60]:

```
import numpy as np

def manhattan_distance(point1, point2):
    point1 = np.asarray(point1)
    point2 = np.asarray(point2)
    distance = np.sum(np.abs(point1 - point2))
    return distance
```

[61]:

```
point1 = [1, 2, 3]
point2 = [4, 5, 6]
distance = manhattan_distance(point1, point2)
print(distance)
```

```
9
```