aiml-lab-7th-sem-1

January 10, 2024

1 1.Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.

[1]:
```python
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)

# Create and train a k-NN classifier with k=3
knn = KNeighborsClassifier(n_neighbors=3).fit(X_train, y_train)

# Predict the labels for the test set
y_pred = knn.predict(X_test)

# Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)

# Print the correct and wrong predictions
for actual, predicted in zip(y_test, y_pred):
    result = "Correct" if actual == predicted else "Wrong"
    print(f"{result} Prediction: Actual = {actual}, Predicted = {predicted}")

# Print summary
print(f"Accuracy: {accuracy * 100:.2f}%")
print(f"Correct Predictions: {sum(y_test == y_pred)}")
print(f"Wrong Predictions: {sum(y_test != y_pred)}")
```

```
Correct Prediction: Actual = 1, Predicted = 1
Correct Prediction: Actual = 0, Predicted = 0
```

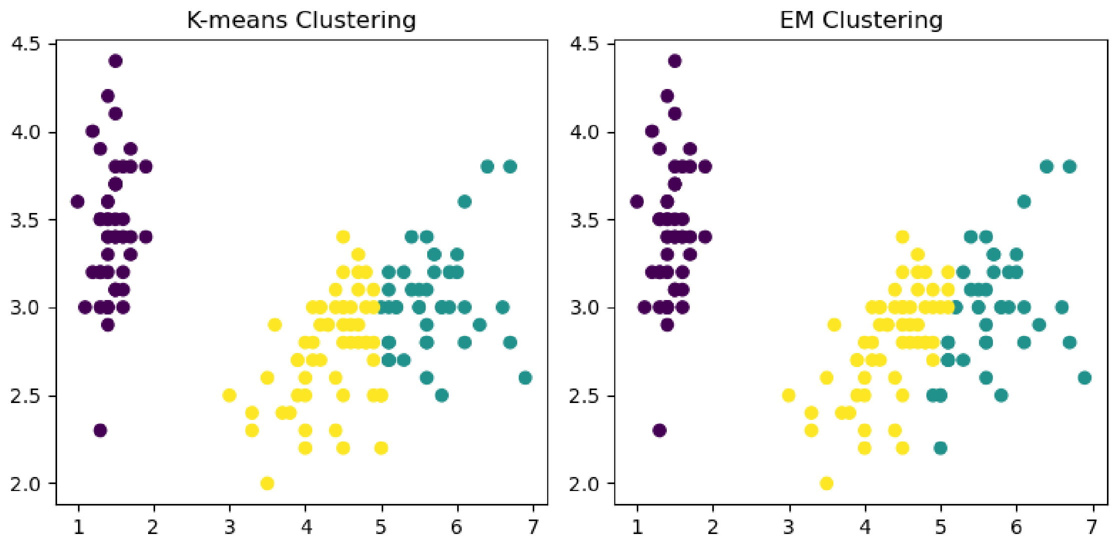
```
Correct Prediction: Actual = 2, Predicted = 2
Correct Prediction: Actual = 1, Predicted = 1
Correct Prediction: Actual = 1, Predicted = 1
Correct Prediction: Actual = 0, Predicted = 0
Correct Prediction: Actual = 1, Predicted = 1
Correct Prediction: Actual = 2, Predicted = 2
Correct Prediction: Actual = 1, Predicted = 1
Correct Prediction: Actual = 1, Predicted = 1
Correct Prediction: Actual = 2, Predicted = 2
Correct Prediction: Actual = 0, Predicted = 0
Correct Prediction: Actual = 0, Predicted = 0
Correct Prediction: Actual = 0, Predicted = 0
Correct Prediction: Actual = 0, Predicted = 0
Correct Prediction: Actual = 1, Predicted = 1
Correct Prediction: Actual = 2, Predicted = 2
Correct Prediction: Actual = 1, Predicted = 1
Correct Prediction: Actual = 1, Predicted = 1
Correct Prediction: Actual = 2, Predicted = 2
Correct Prediction: Actual = 0, Predicted = 0
Correct Prediction: Actual = 2, Predicted = 2
Correct Prediction: Actual = 0, Predicted = 0
Correct Prediction: Actual = 2, Predicted = 2
Correct Prediction: Actual = 2, Predicted = 2
Correct Prediction: Actual = 2, Predicted = 2
Correct Prediction: Actual = 2, Predicted = 2
Correct Prediction: Actual = 2, Predicted = 2
Correct Prediction: Actual = 0, Predicted = 0
Correct Prediction: Actual = 0, Predicted = 0
Correct Prediction: Actual = 0, Predicted = 0
Correct Prediction: Actual = 0, Predicted = 0
Correct Prediction: Actual = 1, Predicted = 1
Correct Prediction: Actual = 0, Predicted = 0
Correct Prediction: Actual = 0, Predicted = 0
Correct Prediction: Actual = 2, Predicted = 2
Correct Prediction: Actual = 1, Predicted = 1
Correct Prediction: Actual = 0, Predicted = 0
Correct Prediction: Actual = 0, Predicted = 0
Correct Prediction: Actual = 0, Predicted = 0
Correct Prediction: Actual = 2, Predicted = 2
Correct Prediction: Actual = 1, Predicted = 1
Correct Prediction: Actual = 1, Predicted = 1
Correct Prediction: Actual = 0, Predicted = 0
Correct Prediction: Actual = 0, Predicted = 0
Accuracy: 100.00%
Correct Predictions: 45
Wrong Predictions: 0
```

2 2.Develop a program to apply K-means algorithm to cluster a set of data stored in .CSV file. Use the same data set for clustering using EM algorithm. Compare the results of these two algorithms and comment on the quality of clustering.

[1]:
```python
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

# Load the Iris dataset
iris = load_iris()
X = iris.data[:, [2, 1]]  # Using Petal Length and Sepal Width for clustering

# Number of clusters
n_clusters = 3

# Apply K-means clustering and EM clustering
kmeans_labels = KMeans(n_clusters=n_clusters, random_state=0).fit_predict(X)
gmm_labels = GaussianMixture(n_components=n_clusters, random_state=0).
 ↪fit_predict(X)

# Plot the results
plt.figure(figsize=(8, 4))
for i, (labels, title) in enumerate(zip([kmeans_labels, gmm_labels], ['K-means␣
 ↪Clustering', 'EM Clustering'])):
    plt.subplot(1, 2, i+1)
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
    plt.title(title)

plt.tight_layout()
plt.show()
```

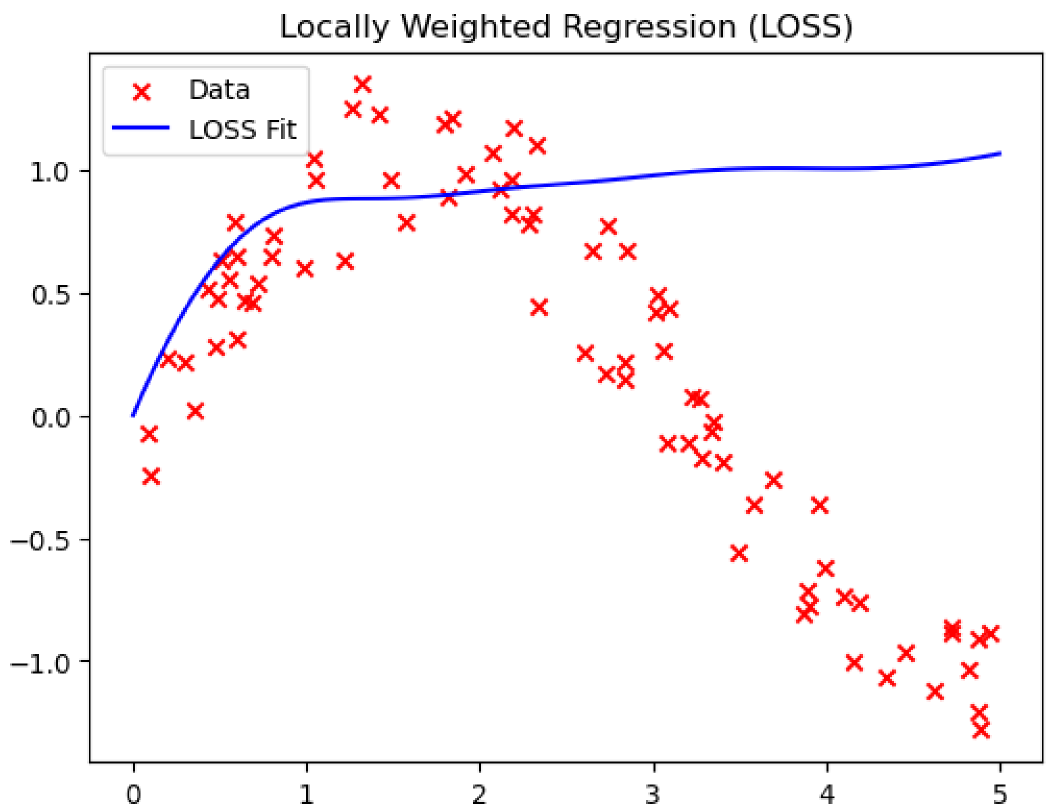
```
C:\Users\Hi\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\Hi\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=1.
  warnings.warn(
C:\Users\Hi\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the
```

```
environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```



3 3.Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

[3]:
```python
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(0)
X = np.sort(5 * np.random.rand(80, 1), axis=0)
y = np.sin(X).ravel() + 0.2 * np.random.randn(80)

def loss(x, X, y, tau=0.5):
    weights = np.exp(-((X - x) ** 2) / (2 * tau ** 2))
    theta = np.sum(X * weights) / np.sum(X ** 2 * weights)
    return theta * x

x_pred = np.linspace(0, 5, 100)
y_pred = [loss(x, X, y) for x in x_pred]  # Use default tau

plt.scatter(X, y, c='r', marker='x', label='Data')
plt.plot(x_pred, y_pred, c='b', label='LOSS Fit')
plt.legend()
plt.title('Locally Weighted Regression (LOSS)')
plt.show()
```



4 4.Build an Artificial Neural Network by implementing the Back-propagation algorithm and test the same using appropriate data sets

[4]:
```
from tensorflow import keras

X = [[0, 0], [0, 1], [1, 0], [1, 1]]
y = [0, 1, 1, 0]

model = keras.Sequential([ keras.layers.Input(shape=(2,)),
                           keras.layers.Dense(4, activation='relu'),
                           keras.layers.Dense(1, activation='sigmoid') ])

model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

model.fit(X, y, epochs=10)

loss, accuracy = model.evaluate(X, y)
```

```
print(f"Loss: {loss}, Accuracy: {accuracy}")
```

```
Epoch 1/10
1/1 [==============================] - 0s 334ms/step - loss: 0.7402 - accuracy:
0.5000
Epoch 2/10
1/1 [==============================] - 0s 18ms/step - loss: 0.7398 - accuracy:
0.5000
Epoch 3/10
1/1 [==============================] - 0s 16ms/step - loss: 0.7394 - accuracy:
0.5000
Epoch 4/10
1/1 [==============================] - 0s 13ms/step - loss: 0.7389 - accuracy:
0.5000
Epoch 5/10
1/1 [==============================] - 0s 0s/step - loss: 0.7385 - accuracy:
0.5000
Epoch 6/10
1/1 [==============================] - 0s 969us/step - loss: 0.7380 - accuracy:
0.5000
Epoch 7/10
1/1 [==============================] - 0s 16ms/step - loss: 0.7376 - accuracy:
0.5000
Epoch 8/10
1/1 [==============================] - 0s 0s/step - loss: 0.7372 - accuracy:
0.5000
Epoch 9/10
1/1 [==============================] - 0s 991us/step - loss: 0.7368 - accuracy:
0.5000
Epoch 10/10
1/1 [==============================] - 0s 17ms/step - loss: 0.7363 - accuracy:
0.5000
1/1 [==============================] - 0s 94ms/step - loss: 0.7359 - accuracy:
0.5000
Loss: 0.7359123229980469, Accuracy: 0.5
```

5 5. Demonstrate Genetic algorithm by taking a suitable data for any simple application.

[11]:
```
import random

def generate_random_string(length):
    sample = '!@#$%^&*()_+=-?><,./:";
↪{\|][}ABCDEFGHIJKLMNOPQRSTUVWXYZqwertyuiopasdfghjklzxcvbnm '
    return ''.join(random.choice(sample) for _ in range(length))
```

```python
def print_hello_world():
    target_string = "HELLO WORLD"
    max_generations = 1000

    current_string = generate_random_string(len(target_string))

    for generation in range(1, max_generations + 1):
        current_string = ''.join(char if current_string[i] == char else␣
 ↪generate_random_string(1) for i, char in enumerate(target_string))
        print(f"Generation {generation}: {current_string}")
        if current_string == target_string:
            break

# Call the function to print the iterations
print_hello_world()
```

```
Generation 1: D%+a\CLUcA]
Generation 2: &/FA^L,"^(=
Generation 3: i@RsUycc>;f
Generation 4: sW\]\"*aoe<
Generation 5: -{C<z#+fvvP
Generation 6: lns;IEuPS%?
Generation 7: KZ+[G;VX@/
Generation 8: TS=_tsoOhR{
Generation 9: cIB($f}OYob
Generation 10: _fVXf:VOea.
Generation 11: LZJ /B^Oaxm
Generation 12: VXFWe #O*F+
Generation 13: vgwFu dO:Y?
Generation 14: _ZSKJ IOngo
Generation 15:  &KT% +O J\
Generation 16: u{#)H ;O};Q
Generation 17: /QC|. zO$k:
Generation 18: {iZbT sOW I
Generation 19: ^pS=@ LO+fH
Generation 20: \D]ZI IO^Ys
Generation 21: }t.!: KOQ|E
Generation 22: ]mMw} IOIi<
Generation 23: G}_VH >O$z]
Generation 24: ")o n !O*H@
Generation 25: kmQQj @OU-{
Generation 26: "I(&S oO.YQ
Generation 27: EPma\ "Ojyj
Generation 28: tv,O| nOGT}
Generation 29: :p+%v uO\GD
Generation 30: =qReZ nOU*D
```

```
Generation 31: uTpgs SO>SD
Generation 32: D=Y\L pOj D
Generation 33: (Ft U >O>BD
Generation 34: F)N%; |O&/D
Generation 35: TE C| dO]-D
Generation 36: PE;bV "OW"D
Generation 37: zEmUf jOl)D
Generation 38: _Ef^. <OJeD
Generation 39: }E>=c .Oc D
Generation 40: "E+H_ *O!oD
Generation 41: _EALY WOvID
Generation 42: |EtLt WO+ D
Generation 43: VE*L{ WOOMD
Generation 44: mEULU WOfnD
Generation 45: qEJLe WOJlD
Generation 46: ,EWLb WOV&D
Generation 47: BE"Lo WOQUD
Generation 48: ;EqLq WO!bD
Generation 49: iE&LL WODYD
Generation 50: qEpLW WO<=D
Generation 51: FEBLh WO}(D
Generation 52: NE>LK WO>,D
Generation 53: ;EbLH WOD>D
Generation 54: ZEJL% WOqHD
Generation 55: ]E#Lb WOiDD
Generation 56: DEdLR WO";D
Generation 57: bEEL- WOz]D
Generation 58: (E@Lc WOD_D
Generation 59: NEFLC WORZD
Generation 60: }E?L- WOR@D
Generation 61: ,EmLO WORwD
Generation 62: NEiLO WORLD
Generation 63: NEaLO WORLD
Generation 64: YEFLO WORLD
Generation 65: hEXLO WORLD
Generation 66: ?ElLO WORLD
Generation 67: vETLO WORLD
Generation 68: !EhLO WORLD
Generation 69: \EDLO WORLD
Generation 70: .E>LO WORLD
Generation 71: IEILO WORLD
Generation 72: ;E LO WORLD
Generation 73: }EMLO WORLD
Generation 74: *EbLO WORLD
Generation 75: /EsLO WORLD
Generation 76: HEVLO WORLD
Generation 77: HEPLO WORLD
Generation 78: HEqLO WORLD
```

```
Generation 79: HEPLO WORLD
Generation 80: HEhLO WORLD
Generation 81: HE[LO WORLD
Generation 82: HEtLO WORLD
Generation 83: HE)LO WORLD
Generation 84: HECLO WORLD
Generation 85: HE]LO WORLD
Generation 86: HEwLO WORLD
Generation 87: HE)LO WORLD
Generation 88: HEJLO WORLD
Generation 89: HEJLO WORLD
Generation 90: HE&LO WORLD
Generation 91: HE<LO WORLD
Generation 92: HE=LO WORLD
Generation 93: HEyLO WORLD
Generation 94: HE%LO WORLD
Generation 95: HEgLO WORLD
Generation 96: HE#LO WORLD
Generation 97: HE+LO WORLD
Generation 98: HEyLO WORLD
Generation 99: HE|LO WORLD
Generation 100: HEmLO WORLD
Generation 101: HE#LO WORLD
Generation 102: HE)LO WORLD
Generation 103: HE#LO WORLD
Generation 104: HE[LO WORLD
Generation 105: HE!LO WORLD
Generation 106: HE:LO WORLD
Generation 107: HEPLO WORLD
Generation 108: HEuLO WORLD
Generation 109: HElLO WORLD
Generation 110: HEqLO WORLD
Generation 111: HEALO WORLD
Generation 112: HE^LO WORLD
Generation 113: HEOLO WORLD
Generation 114: HE=LO WORLD
Generation 115: HEiLO WORLD
Generation 116: HEwLO WORLD
Generation 117: HElLO WORLD
Generation 118: HEXLO WORLD
Generation 119: HEcLO WORLD
Generation 120: HE.LO WORLD
Generation 121: HE&LO WORLD
Generation 122: HEpLO WORLD
Generation 123: HE$LO WORLD
Generation 124: HELLO WORLD
```

6 6.Demonstrate Q learning algorithm with suitable assumption for a problem statement problem: a 2D grid world where an agent needs to find the shortest path to a goal while avoiding obstacles.

[5]:
```python
import numpy as np

env = np.array([[0, 0, 0, 1, 0], [0, 1, 0, 1, 0], [0, 1, 0, 1, 0], [0, 0, 0, 1,␣
↪2]])
Q = np.zeros((20, 4))  # 20 states for a 4x5 grid, 4 actions

def state_to_index(state):
    return state[0] * 5 + state[1]  # 5 columns in the grid

def move(state, action):
    new_state = (max(state[0] - 1, 0), state[1]) if action == 0 else \
                (min(state[0] + 1, 3), state[1]) if action == 1 else \
                (state[0], max(state[1] - 1, 0)) if action == 2 else \
                (state[0], min(state[1] + 1, 4))  # Limit states to valid range
    return new_state
def find_path(Q):
    state = (0, 0)
    path = [state]
    while state != (3, 4):
        action = np.argmax(Q[state_to_index(state)])
        state = move(state, action)
        path.append(state)
    return path


for _ in range(1000):  # Train for 1000 episodes
    state = (0, 0)
    while state != (3, 4):  # Goal state
        action = np.random.choice(4) if np.random.rand() < 0.2 else np.
↪argmax(Q[state_to_index(state)])
        new_state = move(state, action)
        reward = -1 if env[new_state] == 1 else 10 if env[new_state] == 2 else 0
        Q[state_to_index(state)][action] += 0.8 * (reward + 0.95 * np.
↪max(Q[state_to_index(new_state)]) - Q[state_to_index(state)][action])
        state = new_state

optimal_path = find_path(Q)
print("Optimal Path:")
for state in optimal_path:
    print(state)
```

```
Optimal Path:
(0, 0)
(0, 1)
(0, 2)
(1, 2)
(2, 2)
(2, 3)
(2, 4)
(3, 4)
```