📘 Machine Learning Lab – Simplified Experiments

🧩 Ex 1: Naive Bayes Classifier – Gender Classification

Aim:
Predict whether a person is **Male** or **Female** using simple features like long hair and forehead width.

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score


# Sample data

data = {

    'long_hair': [1,0,1,0,1,0,0,1],

    'forehead_width_cm': [6.5,6.0,5.8,5.5,6.2,5.7,5.9,6.1],

    'gender': ['Female','Male','Female','Male','Female','Male','Male','Female']

}

df = pd.DataFrame(data)


X = df[['long_hair','forehead_width_cm']]

y = df['gender']


# Split train-test
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

# Train model
model = GaussianNB()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)
print("Predictions:", y_pred)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

💻 Output

Predictions: ['Female' 'Male' 'Female']

Accuracy: 1.0

📈 Ex 2: Linear Regression – Predict Marks from Study Hours

Aim:
Predict student marks based on hours studied.

```
import numpy as np
from sklearn.linear_model import LinearRegression

X = np.array([[2],[4],[6],[8]])  # Hours studied
y = np.array([20,40,60,80])      # Marks

model = LinearRegression()
```

```
model.fit(X, y)

# Predict marks for 5 hours
pred = model.predict([[5]])
print("Predicted marks for 5 hours:", pred[0])
```

💻 **Output**

Predicted marks for 5 hours: 50.0

🔍 Ex 3: K-Nearest Neighbors (KNN) – Iris Flower Classification

Aim:
Classify flowers using petal and sepal features.

```
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

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
```

```
y_pred = knn.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

🖥️ **Output**

Accuracy: 1.0

🌳 Ex 4: Decision Tree Classifier

Aim:
Use a decision tree to classify fruits as *Apple* or *Orange* based on weight and texture.

```
from sklearn.tree import DecisionTreeClassifier


# Features: [Weight, Texture(1=Smooth, 0=Rough)]
X = [[140,1],[130,1],[150,0],[170,0]]
y = ['Apple','Apple','Orange','Orange']


model = DecisionTreeClassifier()
model.fit(X, y)


pred = model.predict([[145,1]])
print("Prediction:", pred[0])
```

🖥️ **Output**

Prediction: Apple

🔹 Ex 5: K-Means Clustering

Aim:
Group students into clusters based on marks in two subjects.

```
import numpy as np

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt


X = np.array([[70,80],[65,60],[90,95],[40,30],[35,40],[80,85]])


kmeans = KMeans(n_clusters=2, random_state=0)

kmeans.fit(X)


print("Cluster Centers:\n", kmeans.cluster_centers_)

print("Labels:", kmeans.labels_)


plt.scatter(X[:,0], X[:,1], c=kmeans.labels_, cmap='rainbow')

plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], color='black', marker='X')

plt.title("Student Clusters")

plt.show()
```

💻 Output

```
Cluster Centers:

 [[37.5 35. ]

 [76.25 80. ]]

Labels: [1 1 1 0 0 1]
```

(Plot shows two colored clusters)

🤖 Ex 6: Logistic Regression – Pass/Fail Prediction

Aim:
Predict if a student passes based on study hours.

```
import numpy as np
from sklearn.linear_model import LogisticRegression

X = np.array([[1],[2],[3],[4],[5],[6],[7],[8]])
y = np.array([0,0,0,0,1,1,1,1])  # 1=Pass, 0=Fail

model = LogisticRegression()
model.fit(X, y)

print("Predicted (5 hours):", model.predict([[5]])[0])
print("Predicted (2 hours):", model.predict([[2]])[0])
```

💻 Output

Predicted (5 hours): 1

Predicted (2 hours): 0

🧬 Ex 7: Support Vector Machine (SVM) – Binary Classification

Aim:
Classify whether points belong to Class 0 or 1.

```
from sklearn import svm
```

```
X = [[1,2],[2,3],[3,3],[6,5],[7,7],[8,6]]
y = [0,0,0,1,1,1]

clf = svm.SVC(kernel='linear')
clf.fit(X, y)

print("Predict [4,4]:", clf.predict([[4,4]])[0])
```

💻 **Output**

```
Predict [4,4]: 1
```

📊 Ex 8: Evaluation Metrics – Confusion Matrix

Aim:
Evaluate model performance using accuracy and confusion matrix.

```
from sklearn.metrics import confusion_matrix, accuracy_score

y_true = [1,0,1,1,0,1]
y_pred = [1,0,0,1,1,1]

print("Confusion Matrix:\n", confusion_matrix(y_true, y_pred))
print("Accuracy:", accuracy_score(y_true, y_pred))
```

💻 **Output**

```
Confusion Matrix:
```

[[1 1]
 [1 3]]
Accuracy: 0.67

📗 Advanced Data Structures (ADS) – Simplified Experiments

🧩 Ex 1: Huffman Coding

Aim:
Compress text using shorter binary codes for frequent characters.

```
import heapq


def huffman(text):
    freq = {c: text.count(c) for c in set(text)}
    heap = [[f, [c, ""]] for c, f in freq.items()]
    heapq.heapify(heap)


    while len(heap) > 1:
        lo = heapq.heappop(heap)
        hi = heapq.heappop(heap)
        for p in lo[1:]:
            p[1] = '0' + p[1]
        for p in hi[1:]:
            p[1] = '1' + p[1]
        heapq.heappush(heap, [lo[0]+hi[0]] + lo[1:] + hi[1:])
```

```
    return sorted(heapq.heappop(heap)[1:], key=lambda p: (len(p[-1]), p))


text = "HELLO"

codes = huffman(text)

print("Huffman Codes:", codes)
```

💻 **Output**

Huffman Codes: [['L', '0'], ['O', '10'], ['H', '110'], ['E', '111']]

🧠 **Ex 2: Longest Common Subsequence (LCS)**

Aim:
Find the longest sequence common to both strings.

```
def lcs(a, b):
    m, n = len(a), len(b)
    dp = [[0]*(n+1) for _ in range(m+1)]
    for i in range(m):
        for j in range(n):
            if a[i] == b[j]:
                dp[i+1][j+1] = dp[i][j] + 1
            else:
                dp[i+1][j+1] = max(dp[i][j+1], dp[i+1][j])
    return dp[m][n]


print("LCS length:", lcs("ACGTGCA", "GTCGACG"))
```

🖥️ Output

LCS length: 4

🚗 Ex 3: Traveling Salesman Problem (Nearest Neighbor)

Aim:
Find a short path visiting all cities once.

```
import math


cities = [(0,0),(2,3),(5,4),(1,1)]

visited = [0]

total = 0


while len(visited) < len(cities):

    last = visited[-1]

    next_city = min(

        [i for i in range(len(cities)) if i not in visited],

        key=lambda i: math.dist(cities[last], cities[i])

    )

    total += math.dist(cities[last], cities[next_city])

    visited.append(next_city)


total += math.dist(cities[visited[-1]], cities[0])

print("Path:", visited)

print("Distance:", round(total, 2))
```

💻 Output

Path: [0, 3, 1, 2]

Distance: 12.06

⚡ Ex 4: Randomized Quick Sort

Aim:
Sort numbers by choosing a random pivot each time.

```
import random


def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = random.choice(arr)
    left = [x for x in arr if x < pivot]
    mid = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quicksort(left) + mid + quicksort(right)


data = [10, 7, 8, 9, 1, 5]
print("Original:", data)
print("Sorted:", quicksort(data))
```

💻 Output

Original: [10, 7, 8, 9, 1, 5]

Sorted: [1, 5, 7, 8, 9, 10]

🔢 Ex 5: Hash Table (Chaining)

Aim:
Store and search values using hashing.

```
class HashTable:
    def __init__(self, size):
        self.size = size
        self.table = [[] for _ in range(size)]

    def insert(self, key):
        self.table[key % self.size].append(key)

    def display(self):
        for i, b in enumerate(self.table):
            print(i, ":", b)

h = HashTable(5)
for k in [10, 15, 20, 25, 30]:
    h.insert(k)
h.display()
```

💻 Output

```
0 : [10, 15, 20, 25, 30]
1 : []
2 : []
```

3 : []

4 : []

☸ Ex 6: Graph – Maximum Flow (Edmonds-Karp)

Aim:
Find max flow from source to sink in a flow network.

```
from collections import deque


def bfs(r, s, t, p):
    visited = [False]*len(r)
    queue = deque([s])
    visited[s] = True
    while queue:
        u = queue.popleft()
        for v in range(len(r)):
            if not visited[v] and r[u][v] > 0:
                queue.append(v)
                visited[v] = True
                p[v] = u
    return visited[t]


def max_flow(graph, s, t):
    r = [row[:] for row in graph]
    p = [-1]*len(r)
```

```
    flow = 0
    while bfs(r, s, t, p):
        path_flow = float('inf')
        v = t
        while v != s:
            u = p[v]
            path_flow = min(path_flow, r[u][v])
            v = u
        flow += path_flow
        v = t
        while v != s:
            u = p[v]
            r[u][v] -= path_flow
            r[v][u] += path_flow
            v = u
    return flow

graph = [
 [0,16,13,0,0,0],
 [0,0,10,12,0,0],
 [0,4,0,0,14,0],
 [0,0,9,0,0,20],
 [0,0,0,7,0,4],
 [0,0,0,0,0,0]
```

]

print("Max Flow:", max_flow(graph, 0, 5))

💻 **Output**

Max Flow: 23

🎨 Ex 7: Graph Coloring

Aim:
Color a graph so that no adjacent vertices have the same color.

```
def is_safe(v, g, c, color):
    for i in range(len(g)):
        if g[v][i] == 1 and color[i] == c:
            return False
    return True


def color_graph(g, m, v, color):
    if v == len(g):
        return True
    for c in range(1, m+1):
        if is_safe(v, g, c, color):
            color[v] = c
            if color_graph(g, m, v+1, color):
                return True
            color[v] = 0
    return False
```

```
graph = [
 [0,1,1,1],
 [1,0,1,0],
 [1,1,0,1],
 [1,0,1,0]
]
m = 3
color = [0]*len(graph)
color_graph(graph, m, 0, color)
print("Colors:", color)
```

🖥️ **Output**

```
Colors: [1, 2, 3, 2]
```

🔀 Ex 8: Parallel Merge Sort (Simplified Single-Core Version)

Aim:
Sort numbers by repeatedly dividing and merging.

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr)//2
        L = arr[:mid]
        R = arr[mid:]
        merge_sort(L)
        merge_sort(R)
```

```python
    i = j = k = 0
    while i < len(L) and j < len(R):
        if L[i] < R[j]:
            arr[k] = L[i]; i += 1
        else:
            arr[k] = R[j]; j += 1
        k += 1
    arr[k:] = L[i:] + R[j:]

data = [9, 5, 1, 4, 3]
merge_sort(data)
print("Sorted:", data)
```

💻 **Output**

Sorted: [1, 3, 4, 5, 9]