Lab 9 regression

```
import numpy as np
import matplotlib.pyplot as plt

def local_regression(x0, X, Y, tau):
    x0 = [1, x0]
    X = [[1, i] for i in X]
    X = np.asarray(X)
    xw = (X.T) * np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau))
    beta = np.linalg.pinv(xw @ X) @ xw @ Y @ x0
    return beta

def draw(tau):
    prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
    plt.plot(X, Y, 'o', color='black')
    plt.plot(domain, prediction, color='red')
    plt.show()

X = np.linspace(-3, 3, num=1000)
domain = X
Y = np.log(np.abs(X ** 2 - 1) + .5)

draw(10)
draw(0.1)
draw(0.01)
draw(0.001)
```

KNN PROG 8(CSV FILE)

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics

# Load dataset
dataset = pd.read_csv('iris.csv')

# Prepare data
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]

# Split data
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.10)

# Train the KNN classifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(Xtrain, ytrain)

# Make predictions
ypred = classifier.predict(Xtest)

# Evaluate and print results
print(f"Accuracy of the classifier is {metrics.accuracy_score(ytest, ypred)}")
print('\nConfusion Matrix:\n', metrics.confusion_matrix(ytest, ypred))
print('\nClassification Report:\n', metrics.classification_report(ytest, ypred))
```

iris.csv

5.1	3.5	1.4	0.2	Iris-setosa
2 4.9	3.0	1.4	0.2	Iris-setosa
3 4.7	3.2	1.3	0.2	Iris-setosa
4 4.6	3.1	1.5	0.2	Iris-setosa
5 5.0	3.6	1.4	0.2	Iris-setosa
6 5.4	3.9	1.7	0.4	Iris-setosa
7 4.6	3.4	1.4	0.3	Iris-setosa
8 5.0	3.4	1.5	0.2	Iris-setosa
9 4.4	2.9	1.4	0.2	Iris-setosa

Program 7 em algorithm(CSV FILE)

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import matplotlib.pyplot as plt

# read the dataset
dataset = pd.read_csv('iris.csv', names=['slength','swidth','plength','pwidth','species'])

# prepare the data
X = dataset.iloc[:,:-1].values

# set the color map for each species (dictionary)
colormap = {
  'Iris-setosa': 'red',
  'Iris-versicolor': 'green',
  'Iris-virginica': 'blue'
}

# perform K Means clustering and get labels
kmeans = KMeans(n_clusters=3)
kmeans_labels = kmeans.fit_predict(X)

# perform EM algorithm and get labels
em = GaussianMixture(n_components=3)
em_labels = em.fit_predict(X)

# plot the original
plt.subplot(1,3,1)
plt.scatter(X[:, 2], X[:,3], c=dataset['species'].map(colormap))
plt.title('Original')

# plot the kmeans
plt.subplot(1,3,2)
plt.scatter(X[:,2], X[:,3], c=kmeans_labels)
plt.title('K Means Clustering')

# plot the em
plt.subplot(1,3,3)
plt.scatter(X[:,2], X[:,3], c=em_labels)
plt.title('EM Clustering')

# show all the three graphs
plt.show()
```

PROGRAM 6 NAÏVE BAYESON(CSV FILE)

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
data = pd.read_csv('tennis.csv')
print("The first 5 Values of data is :\n", data.head())
X = data.iloc[:, :-1]
print("\nThe First 5 values of the train attributes is\n", X.head())
Y = data.iloc[:, -1]
print("\nThe First 5 values of target values is\n", Y.head())
obj1= LabelEncoder()
X.Outlook = obj1.fit_transform(X.Outlook)
print("\n The Encoded and Transformed Data in Outlook \n",X.Outlook)
obj2 = LabelEncoder()
X.Temperature = obj2.fit_transform(X.Temperature)
obj3 = LabelEncoder()
X.Humidity = obj3.fit_transform(X.Humidity)
obj4 = LabelEncoder()
X.Wind = obj4.fit_transform(X.Wind)
print("\n The Encoded and Transformed Training Examples \n", X.head())
obj5 = LabelEncoder()
Y = obj5.fit_transform(Y)
print("The class Label encoded in numerical form is",Y)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20)
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, Y_train)
from sklearn.metrics import accuracy_score
print("Accuracy is: ", accuracy_score(classifier.predict(X_test), Y_test))
```

PROGRAM 5 ANN

```
import numpy as np

# initial the dataset and target
X = np.array(([2,9],[1,5],[3,6]), dtype=float)
Y = np.array(([92],[86],[89]), dtype=float)

# squish the dataset values to between 0 and 1
X /= 10
Y /= 100

def sigmoid(x):
  return 1 / (1 + np.exp(-x))

# variable initialisation
number_of_iterations = 10000
learning_rate = 0.1
input_nodes = 2
output_nodes = 1
hidden_nodes = 3

# weights and bias initialisation
wh = np.random.uniform(size=(input_nodes, hidden_nodes))
wout = np.random.uniform(size=(hidden_nodes, output_nodes))
bh = np.random.uniform(size=(1, hidden_nodes))
bout = np.random.uniform(size=(1, output_nodes))

# start algorithm
for i in range(number_of_iterations):
  # forward propogation
  hout = sigmoid(np.dot(X, wh) + bh)
  out = sigmoid(np.dot(hout, wout) + bout)

  # backpropogation
  d_out = out * (1 - out) * (Y - out)
  d_hidden = hout * (1 - hout) * np.dot(d_out, wout.T)

  # modify the weights
  wout += learning_rate * np.dot(hout.T, d_out)
  wh += learning_rate * np.dot(X.T, d_hidden)

print('input:\n' + str(X))
print('target:\n' + str(Y))
print('predicted:\n' + str(out))
```

PROGRAM 4 ID3(csv)

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the tennis dataset
tennis_data = pd.read_csv('tennisdata.csv')

# Convert categorical variables to numerical values
tennis_data = pd.get_dummies(tennis_data, columns=['Outlook', 'Temperature',
'Humidity', 'Windy', 'PlayTennis'], drop_first=True)

# Split the dataset into features (X) and target variable (y)
X = tennis_data.drop('PlayTennis_Yes', axis=1)
y = tennis_data['PlayTennis_Yes']

# 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)

# Create and train the Decision Tree model
dt_model = DecisionTreeClassifier(criterion='entropy')
dt_model.fit(X_train, y_train)

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

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Display the Decision Tree
tree_rules = export_text(dt_model, feature_names=X.columns.tolist())
print("\nDecision Tree Rules:")
print(tree_rules)
```

PROGRAM 3 CANDIDATE ELIMINATION

```
import csv

with open("tennis.csv") as f:
    csv_file = csv.reader(f)
    data = list(csv_file)

    specific = data[1][:-1]
    general = [['?' for i in range(len(specific))] for j in range(len(specific))]

    for i in data:
        if i[-1] == "Yes":
            for j in range(len(specific)):
                if i[j] != specific[j]:
                    specific[j] = "?"
                    general[j][j] = "?"

        elif i[-1] == "No":
            for j in range(len(specific)):
                if i[j] != specific[j]:
                    general[j][j] = specific[j]
                else:
                    general[j][j] = "?"

        print("\nStep " + str(data.index(i)+1) + " of Candidate Elimination 
Algorithm")
        print(specific)
        print(general)

        gh = [i for i in general if any(j != '?' for j in i)]


    print("\nFinal Specific hypothesis:\n", specific)
    print("\nFinal General hypothesis:\n", gh)
```

PROG 2

```
class AOStar:
    def __init__(self, heuristic_values, graph, start_node):
        self.heuristic_values = heuristic_values
        self.graph = graph
        self.explored_nodes = set()
        self.solution_graph = {}
        self.start_node = start_node

    def ao_star(self, node):
        print("HEURISTIC VALUES :", self.heuristic_values)
        print("SOLUTION GRAPH :", self.solution_graph)
        print("PROCESSING NODE :", node)
        print("------------------------------------------------------------------------------
-------")

        if node not in self.explored_nodes:
            self.explored_nodes.add(node)
            children = self.expand(node)
            children.sort(key=lambda x: self.heuristic_values[x])
            for child in children:
                self.solution_graph[child] = []
                self.ao_star(child)

    def expand(self, node):
        return self.graph.get(node, [])

# Example usage:
heuristic_values1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7,
'J': 1, 'T': 3}
graph1 = {'A': ['B', 'C'], 'B': ['D'], 'C': ['J'], 'D': ['E', 'F'], 'G': ['I'], 'I': []}

ao_star_instance1 = AOStar(heuristic_values1, graph1, 'A')
ao_star_instance1.ao_star('A')

print("FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE
STARTNODE:", ao_star_instance1.start_node)
print("---------------------------------------------------------")
print(ao_star_instance1.solution_graph)
print("---------------------------------------------------------")
```

PROGRAM 1 A*

```
h = {
    'A': 10,
    'B': 8,
    'C': 5,
    'D': 7,
    'E': 3,
    'F': 6,
    'G': 5,
    'H': 3,
    'I': 1,
    'J': 0
}

Graph_nodes = {
    'A': [('B', 6), ('F', 3)],
    'B': [('C', 3), ('D', 2)],
    'C': [('D', 1), ('E', 5)],
    'D': [('C', 1), ('E', 8)],
    'E': [('I', 5), ('J', 5)],
    'F': [('G', 1), ('H', 7)],
    'G': [('I', 3)],
    'H': [('I', 2)],
    'I': [('E', 5), ('J', 3)]
}

start_node = 'A'
goal_node = 'J'

class Open:
    def __init__(self, node, g, h):
        self.node = node
        self.g = g
        self.h = h
        self.f = g + h

# initialise the open list and closed list arrays
open_list = []
closed_list = []

# insert the start node in open list
start = Open(start_node, 0, h[start_node])
open_list.append(start)
```

```
while len(open_list) > 0:
    # sort open list according to f value
    open_list.sort(key=lambda x:x.f)

    # put 1st node of open list to closed list
    node = open_list[0].node
    closed_list.append(node)

    # check if goal node is reached
    if node == goal_node:
        break

    # add the children of the node to open list and then delete the parent from the
open list
    children = Graph_nodes[node]
    for child in children:
        open_list.append(Open(child[0], child[1] + open_list[0].g, h[child[0]]))
    del open_list[0]

# print the answer (closed list)
print(closed_list)
```

CSV FILE REQUIRED PROGRAM

8,7,6,4,3