4. bayesian networks

import numpy as np

import pandas as pd

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.models import BayesianModel

from pgmpy.inference import VariableElimination

heartDisease = pd.read\_csv('7-dataset.csv')

heartDisease = heartDisease.replace('?',np.nan)

print('Sample instances from the dataset are given below')

print(heartDisease.head())

model=BayesianModel([('age','heartdisease'),('gender','heartdisease'),('exang','heartdisease'),('cp','heartdisease'),('heartdisease','restecg'),('heartdisease','chol')])

print('\nLearning CPD using Maximum likelihood estimators')

model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

print('\n Inferencing with Bayesian Network:')

HeartDiseasetest\_infer = VariableElimination(model)

print('\n 1. Probability of HeartDisease given evidence= restecg')

q1=HeartDiseasetest\_infer.query(variables=['heartdisease'],evidence={'restecg':1})

print(q1)

print('\n 2. Probability of HeartDisease given evidence= cp ')

q2=HeartDiseasetest\_infer.query(variables=['heartdisease'],evidence={'cp':2})

print(q2)

1a. dfs

graph = {

'5' : ['3','7'],

'3' : ['2', '4'],

'7' : ['8'],

'2' : [],

'4' : ['8'],

'8' : []

}

visited =set()

defdfs(visited, graph, node):

if node notin visited:

print (node)

visited.add(node)

for neighbour in graph[node]:

dfs(visited, graph, neighbour)

print("Following is the Depth-First Search")

dfs(visited, graph, '5')

**1b. bfs**

graph = {

'5' : ['3','7'],

'3' : ['2', '4'],

'7' : ['8'],

'2' : [],

'4' : ['8'],

'8' : []

}

visited = []

queue =[]

defbfs(visited, graph, node):

visited.append(node)

queue.append(node)

while queue:

m =queue.pop(0)

print (m, end =" ")

for neighbour in graph[m]:

if neighbour notin visited:

visited.append(neighbour)

queue.append(neighbour)

print("Following is the Breadth-First Search")

bfs(visited, graph, '5')

**2. a\*algorithm**

def aStarAlgo(start\_node, stop\_node):

open\_set = set(start\_node)

closed\_set = set()

g = {}

parents = {}

g[start\_node] = 0

parents[start\_node] = start\_node

while len(open\_set) > 0:

n = None

for v in open\_set:

if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):

n = v

if n == stop\_node or Graph\_nodes[n] == None:

pass

else:

for (m, weight) in get\_neighbors(n):

if m not in open\_set and m not in closed\_set:

open\_set.add(m)

parents[m] = n

g[m] = g[n] + weight

else:

if g[m] > g[n] + weight:

g[m] = g[n] + weight

parents[m] = n

if m in closed\_set:

closed\_set.remove(m)

open\_set.add(m)

if n == None:

print('Path does not exist!')

return None

if n == stop\_node:

path = []

while parents[n] != n:

path.append(n)

n = parents[n]

path.append(start\_node)

path.reverse()

print('Path found: {}'.format(path))

return path

open\_set.remove(n)

closed\_set.add(n)

print('Path does not exist!')

return None

def get\_neighbors(v):

if v in Graph\_nodes:

return Graph\_nodes[v]

else:

return None

def heuristic(n):

H\_dist = {'A': 11,

'B': 6,

'C': 99,

'D': 1,

'E': 7,

'G': 0,

}

return H\_dist[n]

Graph\_nodes = { 'A': [('B', 2), ('E', 3)],

'B': [('C', 1),('G', 9)],

'C': None,

'E': [('D', 6)],

'D': [('G', 1)],

}

aStarAlgo('A', 'G')

**3 navie bayes models**

from sklearn datasets import load\_iris

from sklearn.naive\_bayes import gaussian NB

from sklearn.model\_selection import train\_test.split

from sklearn.matrices import accuracy\_score

iris=load\_iris()

x\_train,x\_test\_y\_train,y\_test=train\_test\_split (iris.data,iris.target,test\_size=0.3.random\_state=4.2)

nb=gaussian NB()

nb.fit(x\_train,y\_train)

y\_pred=nd.predict(x\_test)

accuracy=accuracy\_score(y\_test,y\_pred)

print("accuracy{:2f}%",format(accuracy\*100);

**5a.linear**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

x = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)

y = np.arr([2, 4, 6, 8, 10])

regressor = LinearRegression()

regressor.fit(x, y) y\_pred = regressor.predict(x)

print(‘Coefficients:’, regressor.coef\_)

print(‘Intercept:’, regressor.intercept\_)

plt.scatter(x, y, color = ‘black’)

plt.plot(x, y\_pred, color = ‘blue’, linewidth = 3)

plt.xlabel(‘x’)

plt.ylabel(‘y’)

plt.show()

**5b.logistic**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LogisticRegression

X = np.array([[1, 2], [2, 3], [4, 5], [5, 6]])

y = np.array([0, 0, 1, 1])

classifier = LogisticRegression()

classifier.fit(X, y)

print(‘Coefficient:’, classifier.coef\_)

print(‘Intercept:’, classifier.intercept\_)

xx, yy = np.meshgrid(np.arange(0, 6, 0.01), np.arange(0, 8, 0.01))

z = classifier.predict(np.c\_[xx.ravel(),yy.ravel()])

z = z.reshape(xx.shape)

plt.contourf(xx, yy, z, cmap = plt.cm.RdBu)

plt.scatter(X[:, 0], X[:, 1], c = y, cmap = plt.cm.RdBu\_r, edgecolors = ‘k’)

plt.xlabel(‘Feature 1’)

plt.ylabel(‘Feature 2’)

plt.title(‘Logistic Regression’)

plt.show()

**6a build decision**

from sklearn.datasets import load\_iris

from sklearn.tree import DecisionTreeClassifier

from sklearn.tree import plot\_tree

import matplotlib.pyplot as plt

iris=load\_iris() clf = DecisionTreeClassifier(random\_state = 0)

clf.fit(iris.data,iris.taget)

plot\_tree(clf,filled=True)

plt.show().ylabel(‘y’)

plt.show()

**6b. random forest**

import pandas as pd

from sklearn.ensemble import RandomForestClassifier

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.metrices import accuracy\_score

import matplotlib.pyplot as plt

iris = load\_iris()

df = pd.DataFrame(data = iris.data, columns = iris.feature\_names)

df[‘target’] = iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df[iris.feature\_names], df[‘target’], test\_size = 0.3)

rfc = RandomForestClassifier(n\_estimators = 100, max\_depth = 2, random\_state = 0) rfc.fit(X\_train,y\_train)

y\_pred = rfc.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

importances = rfc.feature\_importances\_ indices = list(range(len(importances)))

plt.bar(indices, importances, color = ‘r’)

plt.xticks(indices, iris.feature\_names, rotation = 90)

plt.title(‘Feature Importance’)

plt.show()

**7. build svm models**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.colors import ListedColormap

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.svm import SVC

from sklearn.metrics import confusion\_matrix

# Load data

data = pd.read\_csv("apples\_and\_oranges.csv")

X = data.iloc[:, 0:2].values

Y = data.iloc[:, 2].values

# Encode target labels

le = LabelEncoder()

Y = le.fit\_transform(Y)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=1)

# Initialize SVM classifier and fit training data

classifier = SVC(kernel='rbf', random\_state=1)

classifier.fit(X\_train, Y\_train)

# Predict classes for test set

Y\_pred = classifier.predict(X\_test)

# Compute accuracy and confusion matrix

cm = confusion\_matrix(Y\_test, Y\_pred)

accuracy = float(cm.diagonal().sum()) / len(Y\_test)

print("Accuracy of SVM for the given dataset: ", accuracy)

# Plot decision boundary and data points

plt.figure(figsize=(7, 7))

X\_set, Y\_set = X\_train, Y\_train

X1, X2 = np.meshgrid(np.arange(start=X\_set[:, 0].min()-1, stop=X\_set[:, 0].max()+1, step=0.01),

np.arange(start=X\_set[:, 1].min()-1, stop=X\_set[:, 1].max()+1, step=0.01))

Z = classifier.predict(np.array([X1.ravel(), X2.ravel()]).T)

Z = Z.reshape(X1.shape)

plt.contourf(X1, X2, Z, alpha=0.75, cmap=ListedColormap(('black', 'white')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(Y\_set)):

plt.scatter(X\_set[Y\_set == j, 0], X\_set[Y\_set == j, 1], color=ListedColormap(('red', 'orange'))(i), label=j)

plt.title('Apples Vs Oranges')

plt.xlabel('Weight in grams')

plt.ylabel('Size in cm')

plt.legend()

plt.show()

**8. ensembling techniques**

from sklearn import datasets

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier, VotingClassifier

from sklearn.svm import SVC

from sklearn.linear\_model import LogisticRegression

# load sample dataset

iris = datasets.load\_iris()

# split dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target,

test\_size=0.3)

# build individual models

svc\_model = SVC(kernel='linear', probability=True)

rf\_model = RandomForestClassifier(n\_estimators=10)

lr\_model = LogisticRegression()

# create ensemble model

ensemble = VotingClassifier(estimators=[('svc', svc\_model), ('rf', rf\_model), ('lr',

lr\_model)],

voting='soft')

# train ensemble model

ensemble.fit(X\_train, y\_train)

# make predictions on test set

y\_pred = ensemble.predict(X\_test)

# print ensemble model accuracy

print("Ensemble Accuracy:", ensemble.score(X\_test, y\_test))

**9.Clustering algorithms**

from sklearn.datasets import make\_blobs

from sklearn.cluster import KMeans, AgglomerativeClustering

import matplotlib.pyplot as plt

# Generate a random dataset with 100 samples and 4 clusters

X, y = make\_blobs(n\_samples=100, centers=4, random\_state=42)

# Create a K-Means clustering object with 4 clusters

kmeans = KMeans(n\_clusters=4, random\_state=42)

# Fit the K-Means model to the dataset

kmeans.fit(X)

# Create a scatter plot of the data colored by K-Means cluster assignment

plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels\_)

plt.title("K-Means Clustering")

plt.show()

# Create a Hierarchical clustering object with 4 clusters

hierarchical = AgglomerativeClustering(n\_clusters=4)

# Fit the Hierarchical model to the dataset

hierarchical.fit(X)

# Create a scatter plot of the data colored by Hierarchical cluster assignment

plt.scatter(X[:, 0], X[:, 1], c=hierarchical.labels\_)

plt.title("Hierarchical Clustering")

plt.show()

**10. em for bayseian networks**

from pgmpy.models import BayesianModel

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.inference import VariableElimination

from pgmpy.factors.discrete import TabularCPD

import numpy as np

# Define the structure of the Bayesian network

model = BayesianModel([('C', 'S'), ('D', 'S')])

# Define the conditional probability distributions (CPDs)

cpd\_c = TabularCPD('C', 2, [[0.5], [0.5]])

cpd\_d = TabularCPD('D', 2, [[0.5], [0.5]])

cpd\_s = TabularCPD('S', 2, [[0.8, 0.6, 0.6, 0.2], [0.2, 0.4, 0.4, 0.8]],

evidence=['C', 'D'], evidence\_card=[2, 2])

# Add the CPDs to the model

model.add\_cpds(cpd\_c, cpd\_d, cpd\_s)

# Create a Maximum Likelihood Estimator and fit the model to some data

data = np.random.randint(low=0, high=2, size=(5000, 2))

mle = MaximumLikelihoodEstimator(model, data)

model\_fit = mle.fit()

# Create a Variable Elimination object to perform inference

infer = VariableElimination(model)

# Perform inference on some observed evidence

query = infer.query(['S'], evidence={'C': 1})

print(query)

**11. Build simple nn models**

import tensorflow as tf

from tensorflow import keras

# Load the MNIST dataset

(x\_train, y\_train), (x\_test, y\_test) = keras.datasets.mnist.load\_data()

# Normalize the input data

x\_train = x\_train / 255.0

x\_test = x\_test / 255.0

# Define the model architecture

model = keras.Sequential([

keras.layers.Flatten(input\_shape=(28, 28)),

keras.layers.Dense(128, activation='relu'),

keras.layers.Dense(10, activation='softmax')

])

# Compile the model

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Train the model

model.fit(x\_train, y\_train, epochs=10, validation\_data=(x\_test, y\_test))

# Evaluate the model

test\_loss, test\_acc = model.evaluate(x\_test, y\_test, verbose=2)

print('Test accuracy:', test\_acc)

**12. build deep learing nn models**

import tensorflow as tf

from tensorflow import keras

# Load the MNIST dataset

(x\_train, y\_train), (x\_test, y\_test) = keras.datasets.mnist.load\_data()

# Normalize the input data

x\_train = x\_train / 255.0

x\_test = x\_test / 255.0

# Define the model architecture

model = keras.Sequential([

keras.layers.Flatten(input\_shape=(28, 28)),

keras.layers.Dense(128, activation='relu'),

keras.layers.Dropout(0.2),

keras.layers.Dense(10)

])

# Compile the model

model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

metrics=['accuracy'])

# Train the model

model.fit(x\_train, y\_train, epochs=10, validation\_data=(x\_test, y\_test))

# Evaluate the model

test\_loss, test\_acc = model.evaluate(x\_test, y\_test, verbose=2)

print('Test accuracy:', test\_acc)