**Subject: ML,DL**

**LAB Assignment**

**1.Find the correlation matrix.**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

# Load Iris dataset

from sklearn.datasets import load\_iris

iris = load\_iris()

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

# Calculate correlation matrix

correlation\_matrix = iris\_df.corr()

print(correlation\_matrix)

**2. Plot the correlation plot on dataset and visualize giving an overview of relationships among data on iris data.**

# Plotting the correlation matrix

plt.figure(figsize=(10, 8))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt='.2f')

plt.title('Correlation Matrix of Iris Dataset')

plt.show()

**3. Analysis of covariance: variance (ANOVA), if data have categorical variables on iris data.**

from scipy import stats

# Perform ANOVA

f\_val, p\_val = stats.f\_oneway(iris\_df['sepal length (cm)'],

iris\_df['sepal width (cm)'],

iris\_df['petal length (cm)'],

iris\_df['petal width (cm)'])

print(f"ANOVA F-value: {f\_val}, P-value: {p\_val}")

**4. Apply linear regression Model techniques to predict the data on any dataset.**

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

from sklearn.linear\_model import LinearRegression

# Prepare data

X = iris\_df[['sepal length (cm)', 'sepal width (cm)']]

y = iris\_df['petal length (cm)']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train Linear Regression model

linear\_model = LinearRegression()

linear\_model.fit(X\_train, y\_train)

# Predictions

predictions = linear\_model.predict(X\_test)

print(predictions)

**5. Apply logical regression Model techniques to predict the data on any dataset.**

from sklearn.linear\_model import LogisticRegression

from sklearn.preprocessing import LabelEncoder

# Prepare data

iris\_df['target'] = LabelEncoder().fit\_transform(iris.target)

X = iris\_df[['sepal length (cm)', 'sepal width (cm)']]

y = iris\_df['target']

# Train Logistic Regression model

logistic\_model = LogisticRegression()

logistic\_model.fit(X, y)

# Predictions

logistic\_predictions = logistic\_model.predict(X)

print(logistic\_predictions)

**6. Clustering algorithms for unsupervised classification.**

from sklearn.cluster import KMeans

# KMeans Clustering

kmeans = KMeans(n\_clusters=3)

iris\_df['Cluster'] = kmeans.fit\_predict(iris\_df[['sepal length (cm)', 'sepal width (cm)']])

**7. Association algorithms for supervised classification on any dataset**

import pandas as pd

from sklearn.datasets import load\_wine

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, confusion\_matrix

# Load Wine dataset

wine = load\_wine()

wine\_df = pd.DataFrame(data=wine.data, columns=wine.feature\_names)

wine\_df['target'] = wine.target

# Prepare data

X = wine\_df.drop('target', axis=1) # Features

y = wine\_df['target'] # Target

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train Random Forest Classifier

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train, y\_train)

# Predictions

y\_pred = rf\_model.predict(X\_test)

# Evaluation

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))# Train Decision Tree

dt\_model = DecisionTreeClassifier()

dt\_model.fit(X, y)

**8. Developing and implementing Decision Tree model on the dataset**

import pandas as pd

from sklearn.datasets import load\_wine

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

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

import matplotlib.pyplot as plt

from sklearn.metrics import classification\_report, confusion\_matrix

# Load Wine dataset

wine = load\_wine()

wine\_df = pd.DataFrame(data=wine.data, columns=wine.feature\_names)

wine\_df['target'] = wine.target

# Prepare features and target variable

X = wine\_df.drop('target', axis=1) # Features

y = wine\_df['target'] # Target variable

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create a Decision Tree Classifier

decision\_tree = DecisionTreeClassifier(random\_state=42)

# Train the model

decision\_tree.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = decision\_tree.predict(X\_test)

# Evaluate the model's performance

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

# Visualize the Decision Tree

plt.figure(figsize=(12, 8))

tree.plot\_tree(decision\_tree, filled=True, feature\_names=X.columns, class\_names=wine.target\_names)

plt.title("Decision Tree for Wine Dataset")

plt.show()

**9. Bayesian classification on any dataset.**

import pandas as pd

from sklearn.datasets import load\_iris

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

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import classification\_report, confusion\_matrix

# Load the Iris dataset

iris = load\_iris()

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

iris\_df['target'] = iris.target

# Prepare features and target variable

X = iris\_df.drop('target', axis=1) # Features

y = iris\_df['target'] # Target variable

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create a Gaussian Naive Bayes Classifier

nb\_model = GaussianNB()

# Train the model

nb\_model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = nb\_model.predict(X\_test)

# Evaluate the model's performance

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

**10. SVM classification on any dataset**

import pandas as pd

from sklearn.datasets import load\_iris

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

from sklearn.svm import SVC

from sklearn.metrics import classification\_report, confusion\_matrix

# Load the Iris dataset

iris = load\_iris()

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

iris\_df['target'] = iris.target

# Prepare features and target variable

X = iris\_df.drop('target', axis=1) # Features

y = iris\_df['target'] # Target variable

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create a Support Vector Classifier

svm\_model = SVC(kernel='linear', random\_state=42)

# Train the model

svm\_model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = svm\_model.predict(X\_test)

# Evaluate the model's performance

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

**11. Text Mining algorithms on unstructured dataset**

import pandas as pd

from sklearn.datasets import fetch\_20newsgroups

from sklearn.feature\_extraction.text import TfidfVectorizer

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

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import classification\_report, confusion\_matrix

# Load the 20 Newsgroups dataset

newsgroups = fetch\_20newsgroups(subset='all', remove=('headers', 'footers', 'quotes'))

# Create a DataFrame

df = pd.DataFrame({'text': newsgroups.data, 'target': newsgroups.target})

# Display the first few rows of the DataFrame

print(df.head())

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df['text'], df['target'], test\_size=0.2, random\_state=42)

# Create a TF-IDF Vectorizer

vectorizer = TfidfVectorizer(stop\_words='english')

# Fit and transform the training data, transform the test data

X\_train\_tfidf = vectorizer.fit\_transform(X\_train)

X\_test\_tfidf = vectorizer.transform(X\_test)

# Create a Multinomial Naive Bayes classifier

nb\_model = MultinomialNB()

# Train the model

nb\_model.fit(X\_train\_tfidf, y\_train)

# Make predictions on the test set

y\_pred = nb\_model.predict(X\_test\_tfidf)

# Evaluate the model's performance

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred, target\_names=newsgroups.target\_names))

**12. Plot the cluster data using python visualizations.**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load\_iris

from sklearn.cluster import KMeans

# Load the Iris dataset

iris = load\_iris()

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

iris\_df['target'] = iris.target

# Define the number of clusters

num\_clusters = 3

# Create a KMeans model

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

# Fit the model to the data

iris\_df['cluster'] = kmeans.fit\_predict(iris\_df[iris.feature\_names])

# Set the figure size

plt.figure(figsize=(10, 6))

# Create a scatter plot

sns.scatterplot(data=iris\_df, x='sepal length (cm)', y='sepal width (cm)', hue='cluster', palette='Set1', style='target', markers=["o", "s", "D"])

# Add centroids to the plot

centroids = kmeans.cluster\_centers\_

plt.scatter(centroids[:, 0], centroids[:, 1], s=300, c='red', marker='X', label='Centroids')

# Add titles and labels

plt.title('K-Means Clustering of Iris Dataset')

plt.xlabel('Sepal Length (cm)')

plt.ylabel('Sepal Width (cm)')

plt.legend()

plt.grid()

plt.show()

**13. Creating & Visualizing Neural Network for the given data. (Use python)**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

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

from sklearn.preprocessing import OneHotEncoder

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.utils import plot\_model

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target.reshape(-1, 1)

# One-hot encode the target variable

encoder = OneHotEncoder(sparse=False)

y\_onehot = encoder.fit\_transform(y)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_onehot, test\_size=0.2, random\_state=42)

# Create a Sequential model

model = Sequential()

model.add(Dense(10, input\_shape=(X\_train.shape[1],), activation='relu')) # Input layer

model.add(Dense(10, activation='relu')) # Hidden layer

model.add(Dense(y\_train.shape[1], activation='softmax')) # Output layer

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the model

history = model.fit(X\_train, y\_train, epochs=100, batch\_size=5, validation\_split=0.2, verbose=1)

# Evaluate the model on the test set

test\_loss,

**14. Recognize optical character using ANN.**

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten

from tensorflow.keras.datasets import mnist

from tensorflow.keras.utils import to\_categorical

# Load the MNIST dataset

(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()

# Normalize the images to the range [0, 1]

X\_train = X\_train.astype('float32') / 255.0

X\_test = X\_test.astype('float32') / 255.0

# One-hot encode the target variable

y\_train = to\_categorical(y\_train, num\_classes=10)

y\_test = to\_categorical(y\_test, num\_classes=10)

# Create a Sequential model

model = Sequential()

model.add(Flatten(input\_shape=(28, 28))) # Flatten the input (28x28 images) to a 1D array (784)

model.add(Dense(128, activation='relu')) # Hidden layer

model.add(Dense(64, activation='relu')) # Hidden layer

model.add(Dense(10, activation='softmax')) # Output layer

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Train the model

history = model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_split=0.2, verbose=1)

# Evaluate the model on the test set

test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test)

print(f"Test Loss: {test\_loss:.4f}, Test Accuracy: {test\_accuracy:.4f}")

# Plot training & validation accuracy values

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

# Plot training & validation loss values

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.tight\_layout()

plt.show()

# Make predictions on the test set

predictions = model.predict(X\_test)

# Function to visualize predictions

def visualize\_predictions(X, y\_true, y\_pred, num\_images=10):

plt.figure(figsize=(15, 5))

for i in range(num\_images):

plt.subplot(2, 5, i + 1)

plt.imshow(X[i], cmap='gray')

plt.title(f'True: {np.argmax(y\_true[i])}, Pred: {np.argmax(y\_pred[i])}')

plt.axis('off')

plt.show()

# Visualize the first 10 predictions

visualize\_predictions(X\_test, y\_test, predictions)

**15. Write a program to implement CNN**

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.callbacks import EarlyStopping

# Load the CIFAR-10 dataset

(X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data()

# Normalize the images to the range [0, 1]

X\_train = X\_train.astype('float32') / 255.0

X\_test = X\_test.astype('float32') / 255.0

# One-hot encode the target variable

y\_train = to\_categorical(y\_train, num\_classes=10)

y\_test = to\_categorical(y\_test, num\_classes=10)

# Create a Sequential model

model = Sequential()

# Add convolutional layers

model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(128, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

# Flatten the output from the convolutional layers

model.add(Flatten())

# Add fully connected layers

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5)) # Dropout for regularization

model.add(Dense(10, activation='softmax')) # Output layer

# Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Set early stopping to prevent overfitting

early\_stopping = EarlyStopping(monitor='val\_loss', patience=3, restore\_best\_weights=True)

# Train the model

history = model.fit(X\_train, y\_train, epochs=20, batch\_size=64, validation\_split=0.2, callbacks=[early\_stopping], verbose=1)

# Evaluate the model on the test set

test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test)

print(f"Test Loss: {test\_loss:.4f}, Test Accuracy: {test\_accuracy:.4f}")

# Plot training & validation accuracy values

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

# Plot training & validation loss values

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.tight\_layout()

plt.show()

**16. Write a program to implement RNN**

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import SimpleRNN, Dense, Embedding

# Sample text data

text = "hello world this is a simple example of rnn implementation"

# Create a set of unique characters

chars = sorted(list(set(text)))

char\_to\_index = {ch: i for i, ch in enumerate(chars)}

index\_to\_char = {i: ch for i, ch in enumerate(chars)}

# Prepare the input and output sequences

maxlen = 5 # Length of the input sequences

step = 1 # Step size

sentences = []

next\_chars = []

for i in range(0, len(text) - maxlen, step):

sentences.append(text[i: i + maxlen])

next\_chars.append(text[i + maxlen])

# Vectorization

X = np.zeros((len(sentences), maxlen, len(chars)), dtype=np.bool)

y = np.zeros((len(sentences), len(chars)), dtype=np.bool)

for i, sentence in enumerate(sentences):

for t, char in enumerate(sentence):

X[i, t, char\_to\_index[char]] = 1

y[i, char\_to\_index[next\_chars[i]]] = 1

# Create a Sequential model

model = Sequential()

# Add an embedding layer

model.add( Embedding(len(chars), 10, input\_length=maxlen))

# Add a simple RNN layer

model.add(SimpleRNN(128, activation='relu'))

# Add an output layer

model.add(Dense(len(chars), activation='softmax'))

# Compile the model

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model

model.fit(X, y, batch\_size=1, epochs=100)

def generate\_text(seed, length=50):

generated = seed

for i in range(length):

x\_pred = np.zeros((1, maxlen, len(chars)))

for t, char in enumerate(seed):

x\_pred[0, t, char\_to\_index[char]] = 1

preds = model.predict(x\_pred, verbose=0)[0]

next\_index = np.argmax(preds)

next\_char = index\_to\_char[next\_index]

generated += next\_char

seed = seed[1:] + next\_char # Update seed by removing the first character and adding the new one

return generated

# Generate text

seed\_text = "hello"

generated\_text = generate\_text(seed\_text)

print("Generated text:", generated\_text)

**17. Write a program to implement GAN**

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential, Model

from tensorflow.keras.layers import Dense, Reshape, Flatten, Dropout, BatchNormalization, LeakyReLU

from tensorflow.keras.optimizers import Adam

# Load the MNIST dataset

(X\_train, \_), (\_, \_) = mnist.load\_data()

# Normalize the images to the range [-1, 1]

X\_train = X\_train.astype(np.float32) / 255.0

X\_train = (X\_train - 0.5) \* 2.0 # Scale to [-1, 1]

# Reshape the data to fit the model

X\_train = np.expand\_dims(X\_train, axis=-1)

def build\_generator(z\_dim):

model = Sequential()

model.add(Dense(256, input\_dim=z\_dim))

model.add(LeakyReLU(alpha=0.2))

model.add(BatchNormalization(momentum=0.8))

model.add(Dense(512))

model.add(LeakyReLU(alpha=0.2))

model.add(BatchNormalization(momentum=0.8))

model.add(Dense(1024))

model.add(LeakyReLU(alpha=0.2))

model.add(BatchNormalization(momentum=0.8))

model.add(Dense(28 \* 28 \* 1, activation='tanh')) # Output layer

model.add(Reshape((28, 28, 1)))

return model

z\_dim = 100 # Dimension of the noise vector

generator = build\_generator(z\_dim)

def build\_discriminator(img\_shape):

model = Sequential()

model.add(Flatten(input\_shape=img\_shape))

model.add(Dense(512))

model.add(LeakyReLU(alpha=0.2))

model.add(Dropout(0.3))

model.add(Dense(256))

model.add(LeakyReLU(alpha=0.2))

model.add(Dropout(0.3))

model.add(Dense(1, activation='sigmoid')) # Output layer

return model

img\_shape = (28, 28, 1)

discriminator = build\_discriminator(img\_shape)

# Compile the discriminator

discriminator.compile(loss='binary\_crossentropy', optimizer=Adam(0.0002, 0.5), metrics=['accuracy'])

# Create the GAN model

discriminator.trainable = False # Freeze the discriminator when training the GAN

z = tf.keras.Input(shape=(z\_dim,))

img = generator(z)

validity = discriminator(img)

gan = Model(z, validity)

gan.compile(loss='binary\_crossentropy', optimizer=Adam(0.0002, 0.5))

def train\_gan(epochs, batch\_size=128):

generated\_images = []

for epoch in range(epochs):

idx = np.random.randint(0, X\_train.shape[0], batch\_size)

real\_images = X\_train[idx]

noise = np.random.normal(0, 1, (batch\_size, z\_dim))

fake\_images = generator.predict(noise)

d\_loss\_real = discriminator.train\_on\_batch(real\_images, np.ones((batch\_size, 1)))

d\_loss\_fake = discriminator.train\_on\_batch(fake\_images, np.zeros((batch\_size, 1)))

d\_loss = 0.5 \* np.add(d\_loss\_real, d\_loss\_fake)

noise = np.random.normal(0, 1, (batch\_size, z\_dim))

g\_loss = gan.train\_on\_batch(noise, np.ones((batch\_size, 1)))

if epoch % 100 == 0:

print(f"{epoch} [D loss: {d\_loss[0]:.4f}, acc.: {100 \* d\_loss[1]:.2f}%] [G loss: {g\_loss:.4f}]")

if epoch % 1000 == 0:

generated\_images.append(fake\_images)

return generated\_images

# Train the GAN

generated\_images = train\_gan(epochs=10000, batch\_size=128)

# Visualize the generated images

plt.figure(figsize=(10, 10))

for i, img in enumerate(generated\_images):

plt.subplot(1, len(generated\_images), i + 1)

plt.imshow(img[0, :, :, 0], cmap

**18. Web scraping experiments (by using tools)**

**Beautiful Soup (Python)**

pip install beautifulsoup4 requests

import requests

from bs4 import BeautifulSoup

# URL to scrape

url = 'https://example.com'

# Make a request to the website

response = requests.get(url)

# Parse the HTML content

soup = BeautifulSoup(response.content, 'html.parser')

# Find all headings (h1, h2, etc.)

headings = soup.find\_all(['h1', 'h2', 'h3'])

for heading in headings:

print(heading.text)

**Selenium**

pip install selenium

from selenium import webdriver

from selenium.webdriver.common.by import By

# Set up the WebDriver (e.g., Chrome)

driver = webdriver.Chrome()

# Open a webpage

driver.get('https://example.com')

# Extract data

headings = driver.find\_elements(By.TAG\_NAME, 'h1')

for heading in headings:

print(heading.text)

# Close the driver

driver.quit()

**Puppeteer (Node.js)**

npm install puppeteer

const puppeteer = require('puppeteer');

(async () => {

const browser = await puppeteer.launch();

const page = await browser.newPage();

await page.goto('https://example.com');

// Extract data

const headings = await page.evaluate(() => {

return Array.from(document.querySelectorAll('h1, h2, h3')).map(h => h.innerText);

});

console.log(headings);

await browser.close();

})();