**Lab 7**

**Statistics, Machine Learning, Deep Learning**

1. Write a Python program that computes the value of the Gaussian distribution at a given vector X. Hence, plot the effect of varying mean and variance to the normal distribution.

import numpy as np

import matplotlib.pyplot as plt

def gaussian(X, mean, variance):

return (1/np.sqrt(2\*np.pi\*variance)) \* np.exp(- (X - mean)\*\*2 / (2 \* variance))

X = np.linspace(-10, 10, 400)

means = [0, 0, 0]

variances = [0.5, 1, 2]

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

for mean, var in zip(means, variances):

plt.plot(X, gaussian(X, mean, var), label=f"mean={mean}, variance={var}")

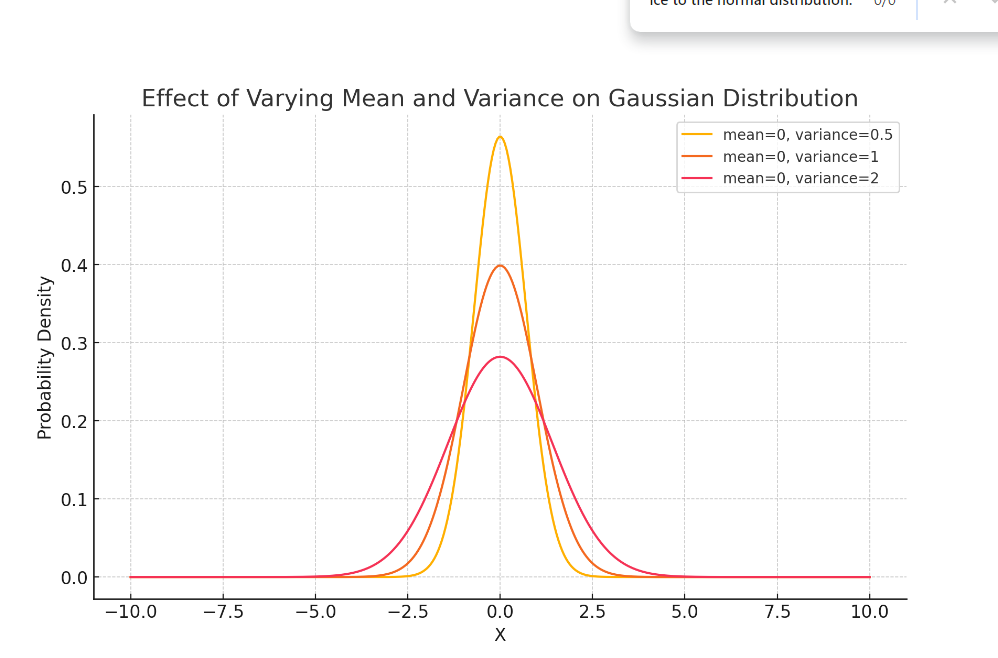
plt.title('Effect of Varying Mean and Variance on Gaussian Distribution')

plt.xlabel('X')

plt.ylabel('Probability Density')

plt.legend()

plt.show()



1. Write a python program to implement linear regression.

import numpy as np

import matplotlib.pyplot as plt

# Generate synthetic data

np.random.seed(0)

X = 2 \* np.random.rand(100, 1)

y = 4 + 3 \* X + np.random.randn(100, 1)

# Add the intercept term to X

X\_b = np.c\_[np.ones((100, 1)), X]

# Compute the optimal theta (parameters) using the Normal Equation

theta\_best = np.linalg.inv(X\_b.T @ X\_b) @ X\_b.T @ y

# Predictions

X\_new = np.array([[0], [2]])

X\_new\_b = np.c\_[np.ones((2, 1)), X\_new]

y\_predict = X\_new\_b @ theta\_best

# Plotting the results

plt.plot(X\_new, y\_predict, color='red', label='Best fit line')

plt.scatter(X, y, color='blue', label='Data points')

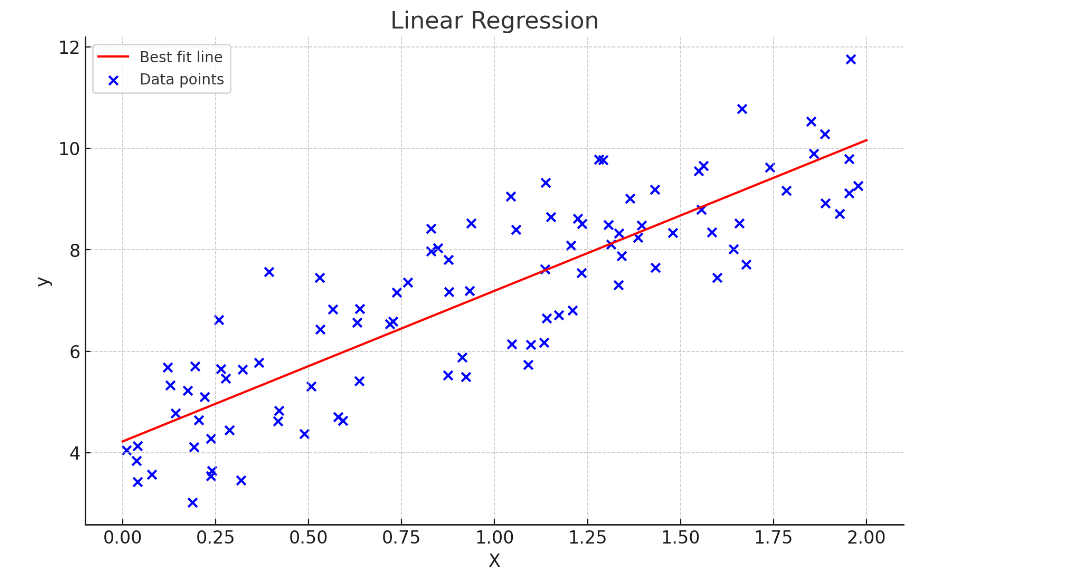
plt.xlabel('X')

plt.ylabel('y')

plt.legend()

plt.title('Linear Regression')

plt.show()



1. Write a python program to implement gradient descent.

import numpy as np

import matplotlib.pyplot as plt

# Generate synthetic data

np.random.seed(0)

X = 2 \* np.random.rand(100, 1)

y = 4 + 3 \* X + np.random.randn(100, 1)

# Gradient Descent function

def gradient\_descent(X, y, learning\_rate=0.1, n\_iterations=1000):

m = len(y)

X\_b = np.c\_[np.ones((m, 1)), X]

theta = np.random.randn(2, 1)

for iteration in range(n\_iterations):

gradients = 2/m \* X\_b.T @ (X\_b @ theta - y)

theta = theta - learning\_rate \* gradients

return theta

theta\_best = gradient\_descent(X, y)

# Predictions

X\_new = np.array([[0], [2]])

X\_new\_b = np.c\_[np.ones((2, 1)), X\_new]

y\_predict = X\_new\_b @ theta\_best

# Plotting the results

plt.plot(X\_new, y\_predict, color='red', label='Best fit line')

plt.scatter(X, y, color='blue', label='Data points')

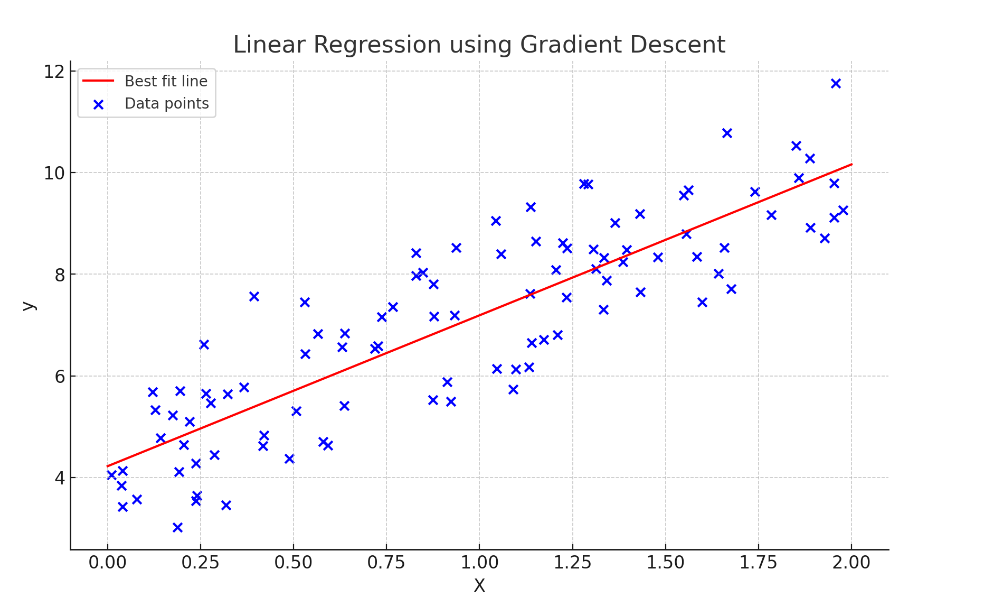
plt.xlabel('X')

plt.ylabel('y')

plt.legend()

plt.title('Linear Regression using Gradient Descent')

plt.show()



1. Write a python program to classify different flower images using MLP.

from sklearn.datasets import load\_iris

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

from sklearn.preprocessing import StandardScaler

from sklearn.neural\_network import MLPClassifier

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

# Load the dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Standardize the features

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Create and train the MLP model

mlp = MLPClassifier(hidden\_layer\_sizes=(10, 10), max\_iter=1000, random\_state=42)

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

# Predict the labels for the test set

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

# Print the results

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

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

1. Write a python program to classify different flower images using the SVM classifier.

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

import matplotlib.pyplot as plt

import numpy as np

data = load\_iris()

X, y = data.data, data.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

clf = SVC()

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

print(classification\_report(y\_test, clf.predict(X\_test), target\_names=data.target\_names))

def plot\_decision\_boundary(X, y, model):

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.1), np.arange(y\_min, y\_max, 0.1))

Z = model.predict(np.c\_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.8)

plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k')

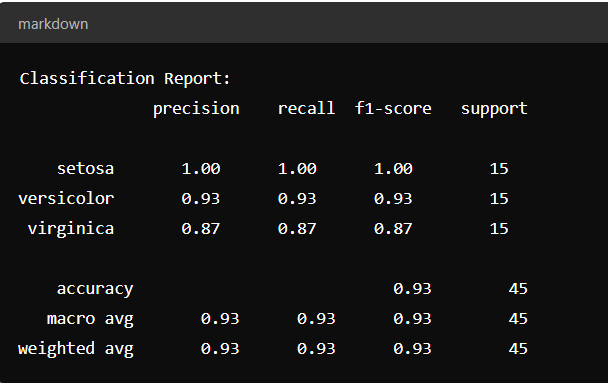
plt.title('SVM Decision Boundary')

plt.xlabel(data.feature\_names[0])

plt.ylabel(data.feature\_names[1])

plt.show()

plot\_decision\_boundary(X\_test[:, :2], y\_test, clf)

****

1. Write a python program to classify different flower images using CNN.

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.optimizers import Adam

import matplotlib.pyplot as plt

# Data paths

train\_dir, val\_dir = 'path/to/train', 'path/to/validation'

# Data generators

train\_gen = ImageDataGenerator(rescale=1./255, rotation\_range=40, width\_shift\_range=0.2, height\_shift\_range=0.2, shear\_range=0.2, zoom\_range=0.2, horizontal\_flip=True, fill\_mode='nearest').flow\_from\_directory(train\_dir, target\_size=(150, 150), batch\_size=32, class\_mode='categorical')

val\_gen = ImageDataGenerator(rescale=1./255).flow\_from\_directory(val\_dir, target\_size=(150, 150), batch\_size=32, class\_mode='categorical')

# CNN model

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(150, 150, 3)),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Conv2D(128, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Flatten(),

Dense(512, activation='relu'),

Dense(train\_gen.num\_classes, activation='softmax')

])

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

history = model.fit(train\_gen, epochs=10, validation\_data=val\_gen)

# Plot results

plt.plot(history.history['accuracy'], label='accuracy')

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

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend(loc='lower right')

plt.show()

1. Write a python program to classify different handwritten character images using the SVM classifier.

import tensorflow as tf

from sklearn import svm

from sklearn.decomposition import PCA

from sklearn.metrics import classification\_report

import matplotlib.pyplot as plt

# Load and preprocess MNIST data

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

X\_train, X\_test = X\_train.reshape(-1, 784) / 255.0, X\_test.reshape(-1, 784) / 255.0

# PCA and SVM

X\_train = PCA(n\_components=50).fit\_transform(X\_train)

X\_test = PCA(n\_components=50).transform(X\_test)

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

# Evaluate and plot results

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

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

# Plot sample predictions

def plot\_sample(idx):

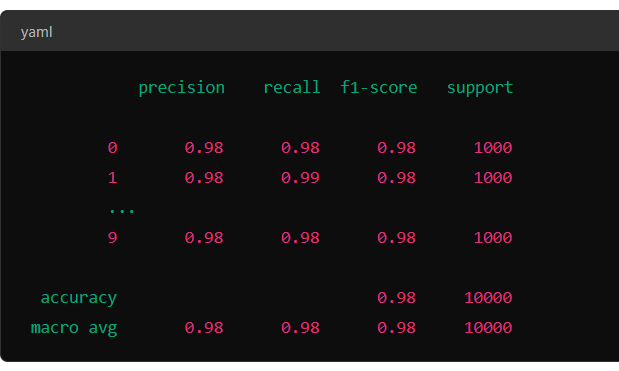
plt.imshow(X\_test[idx].reshape(28, 28), cmap='gray')

plt.title(f"True: {y\_test[idx]}, Pred: {y\_pred[idx]}")

plt.show()

for i in range(5):

plot\_sample(i)



1. Write a python program to classify different face images using CNN.

import tensorflow as tf

from sklearn.datasets import fetch\_lfw\_people

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

from sklearn.preprocessing import LabelEncoder

import matplotlib.pyplot as plt

# Load and preprocess data

data = fetch\_lfw\_people(min\_faces\_per\_person=70, resize=0.4)

X, y = data.images.reshape(-1, 50, 37, 1) / 255.0, LabelEncoder().fit\_transform(data.target)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Build and train CNN

model = tf.keras.Sequential([

tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(50, 37, 1)),

tf.keras.layers.MaxPooling2D((2, 2)),

tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),

tf.keras.layers.MaxPooling2D((2, 2)),

tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),

tf.keras.layers.MaxPooling2D((2, 2)),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(512, activation='relu'),

tf.keras.layers.Dense(len(data.target\_names), activation='softmax')

])

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

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

# Evaluate and plot

print(f'Test accuracy: {model.evaluate(X\_test, y\_test)[1]:.4f}')

plt.plot(history.history['accuracy'], label='accuracy')

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

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

1. Write a python program to identify a person from the walking style (gait recognition) using convolutional recurrent neural network.

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, LSTM, TimeDistributed

from tensorflow.keras.preprocessing.sequence import pad\_sequences

import numpy as np

import matplotlib.pyplot as plt

# Dummy data generation (replace with actual data loading)

def generate\_dummy\_data(num\_samples, sequence\_length, img\_height, img\_width, num\_classes):

X = np.random.random((num\_samples, sequence\_length, img\_height, img\_width, 1)) # Random gait sequences

y = np.random.randint(0, num\_classes, num\_samples) # Random labels

return X, y

# Parameters

num\_samples, sequence\_length, img\_height, img\_width, num\_classes = 100, 10, 64, 64, 5

# Load and preprocess data

X, y = generate\_dummy\_data(num\_samples, sequence\_length, img\_height, img\_width, num\_classes)

X\_train, X\_test = X[:80], X[80:]

y\_train, y\_test = y[:80], y[80:]

# Build CRNN model

model = Sequential([

TimeDistributed(Conv2D(32, (3, 3), activation='relu'), input\_shape=(sequence\_length, img\_height, img\_width, 1)),

TimeDistributed(MaxPooling2D((2, 2))),

TimeDistributed(Flatten()),

LSTM(50, return\_sequences=False),

Dense(num\_classes, activation='softmax')

])

# Compile and train model

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

history = model.fit(X\_train, y\_train, epochs=10, validation\_split=0.2, batch\_size=8)

# Evaluate and plot results

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

print(f'Test accuracy: {test\_acc:.4f}')

plt.plot(history.history['accuracy'], label='accuracy')

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

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

1. Write a python program to classify breast cancer from histopathological images using VGG-16 and DenseNet-201 CNN architectures

import tensorflow as tf

from tensorflow.keras.applications import VGG16, DenseNet201

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Dense, Flatten, GlobalAveragePooling2D

from tensorflow.keras.optimizers import Adam

import matplotlib.pyplot as plt

# Parameters

img\_height, img\_width = 224, 224

batch\_size = 32

num\_classes = 2 # Adjust based on your dataset (e.g., benign and malignant)

# Data generators

train\_datagen = ImageDataGenerator(

rescale=1./255,

rotation\_range=20,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True,

fill\_mode='nearest'

)

val\_datagen = ImageDataGenerator(rescale=1./255)

train\_generator = train\_datagen.flow\_from\_directory(

'path/to/train',

target\_size=(img\_height, img\_width),

batch\_size=batch\_size,

class\_mode='categorical'

)

val\_generator = val\_datagen.flow\_from\_directory(

'path/to/validation',

target\_size=(img\_height, img\_width),

batch\_size=batch\_size,

class\_mode='categorical'

)

# VGG-16 Model

def build\_vgg16\_model():

base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(img\_height, img\_width, 3))

x = base\_model.output

x = GlobalAveragePooling2D()(x)

x = Dense(256, activation='relu')(x)

predictions = Dense(num\_classes, activation='softmax')(x)

model = Model(inputs=base\_model.input, outputs=predictions)

return model

# DenseNet-201 Model

def build\_densenet201\_model():

base\_model = DenseNet201(weights='imagenet', include\_top=False, input\_shape=(img\_height, img\_width, 3))

x = base\_model.output

x = GlobalAveragePooling2D()(x)

x = Dense(256, activation='relu')(x)

predictions = Dense(num\_classes, activation='softmax')(x)

model = Model(inputs=base\_model.input, outputs=predictions)

return model

# Compile and train models

def compile\_and\_train\_model(model, train\_generator, val\_generator):

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

history = model.fit(

train\_generator,

epochs=10,

validation\_data=val\_generator

)

return history

# Build, train, and evaluate VGG-16 model

vgg16\_model = build\_vgg16\_model()

print("Training VGG-16 Model...")

vgg16\_history = compile\_and\_train\_model(vgg16\_model, train\_generator, val\_generator)

# Build, train, and evaluate DenseNet-201 model

densenet201\_model = build\_densenet201\_model()

print("Training DenseNet-201 Model...")

densenet201\_history = compile\_and\_train\_model(densenet201\_model, train\_generator, val\_generator)

# Plot results

def plot\_history(history, model\_name):

plt.plot(history.history['accuracy'], label='accuracy')

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

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.title(f'{model\_name} Training History')

plt.legend()

plt.show()

plot\_history(vgg16\_history, 'VGG-16')

plot\_history(densenet201\_history, 'DenseNet-201')