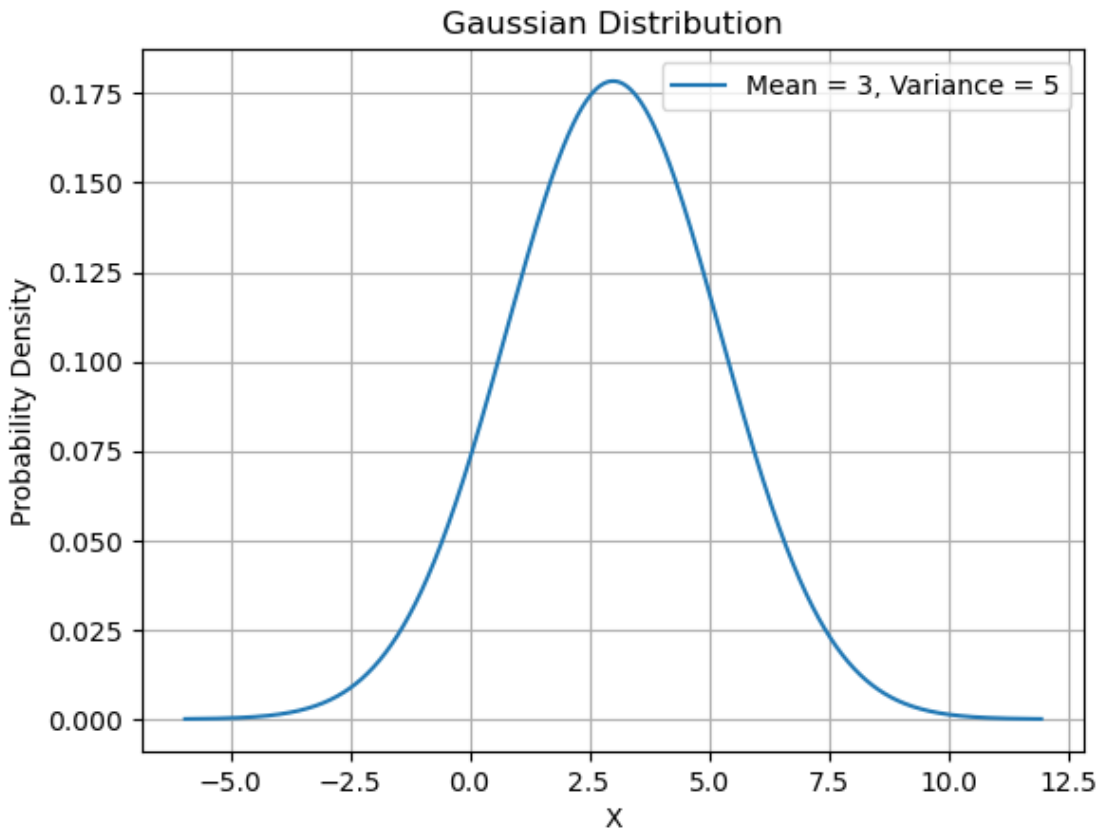
Assignment No. 7

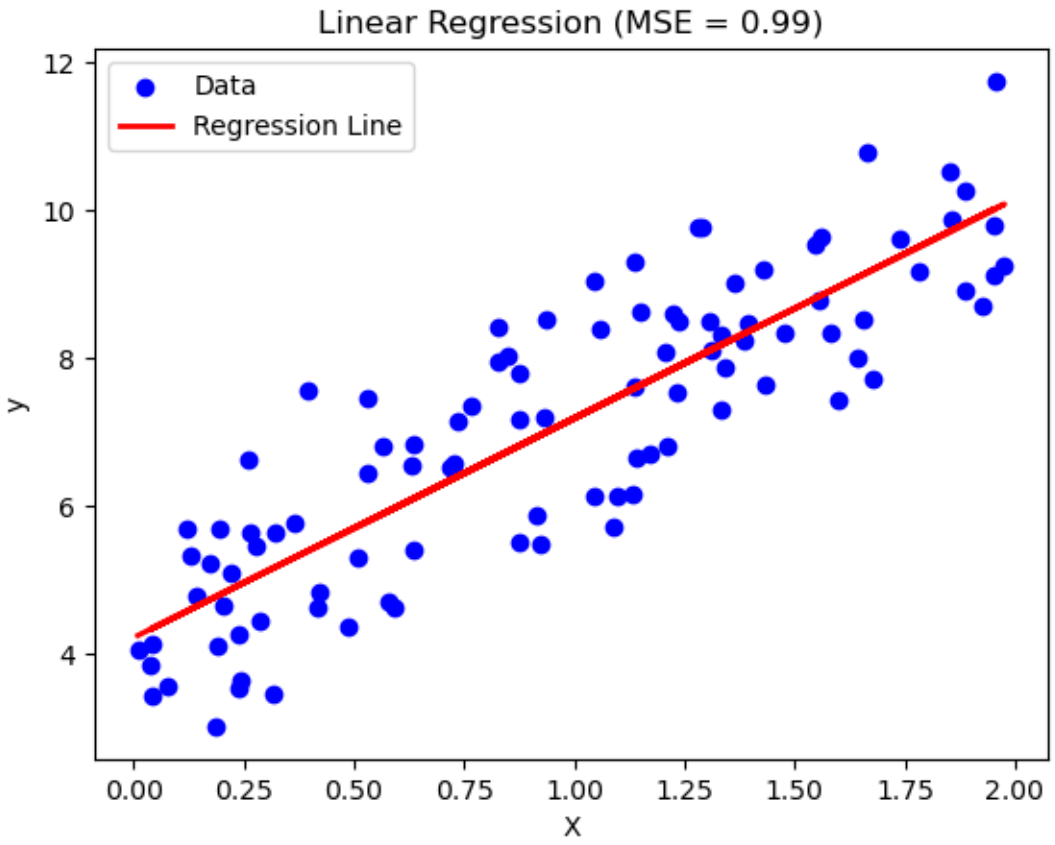
September 4, 2024

[5]:
```
#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
from scipy.stats import norm
def plot_gaussian(mean, variance):
    x = np.linspace(mean - 4*np.sqrt(variance), mean + 4*np.sqrt(variance),␣
 ↪1000)
    y = norm.pdf(x, mean, np.sqrt(variance))
    plt.plot(x, y, label=f'Mean = {mean}, Variance = {variance}')
    plt.xlabel('X')
    plt.ylabel('Probability Density')
    plt.title('Gaussian Distribution')
    plt.legend()
    plt.grid(True)
    plt.show()
plot_gaussian(mean=3, variance=5)
```



[6]:
```python
#2. Write a python program to implement linear regression.
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
np.random.seed(0)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)
# Create and fit the model
model = LinearRegression()
model.fit(X, y)
# Predict and evaluate
y_pred = model.predict(X)
mse = mean_squared_error(y, y_pred)
# Plot
plt.scatter(X, y, color='blue', label='Data')
plt.plot(X, y_pred, color='red', linewidth=2, label='Regression Line')
plt.xlabel('X')
plt.ylabel('y')
plt.title(f'Linear Regression (MSE = {mse:.2f})')
```

```
plt.legend()
plt.show()
```



[7]:
```
#3. Write a python program to implement gradient descent.
def gradient_descent(learning_rate, num_iterations):
    x = float(input("Enter the initial value of x: "))
    for i in range(num_iterations):
        grad = 2 * x  # Gradient of f(x) = x^2 is f'(x) = 2x
        x = x - learning_rate * grad
        print(f"Iteration {i+1}: x = {x}, f(x) = {x**2}")
    return x
learning_rate = float(input("Enter the learning rate: "))
num_iterations = int(input("Enter the number of iterations: "))
final_x = gradient_descent(learning_rate, num_iterations)
print(f"Final value of x after {num_iterations} iterations is {final_x}")
```

```
Enter the learning rate:  2
Enter the number of iterations:  7
Enter the initial value of x:  1

Iteration 1: x = -3.0, f(x) = 9.0
```

```
Iteration 2: x = 9.0, f(x) = 81.0
Iteration 3: x = -27.0, f(x) = 729.0
Iteration 4: x = 81.0, f(x) = 6561.0
Iteration 5: x = -243.0, f(x) = 59049.0
Iteration 6: x = 729.0, f(x) = 531441.0
Iteration 7: x = -2187.0, f(x) = 4782969.0
Final value of x after 7 iterations is -2187.0
```

[8]:
```python
#4. 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.neural_network import MLPClassifier
from sklearn.metrics 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=42)
mlp = MLPClassifier(hidden_layer_sizes=(10,), max_iter=1000)
mlp.fit(X_train, y_train)
y_pred = mlp.predict(X_test)
print("MLP accuracy:", accuracy_score(y_test, y_pred))
```

```
MLP accuracy: 1.0
```

[9]:
```python
#5. 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 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=42)
svm = SVC(kernel='linear')
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
print("SVM accuracy:", accuracy_score(y_test, y_pred))
```

```
SVM accuracy: 1.0
```

[10]:
```python
#6. Write a python program to classify different flower images using CNN.
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
```

```
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))
print("CNN test accuracy:", model.evaluate(X_test, y_test)[1])
```

```
C:\Users\User\anaconda3\Lib\site-
packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/10
1563/1563             11s 6ms/step -
accuracy: 0.3163 - loss: 3.0629 - val_accuracy: 0.5062 - val_loss: 1.3742
Epoch 2/10
1563/1563             9s 6ms/step -
accuracy: 0.5201 - loss: 1.3407 - val_accuracy: 0.5567 - val_loss: 1.2713
Epoch 3/10
1563/1563             9s 6ms/step -
accuracy: 0.5864 - loss: 1.1854 - val_accuracy: 0.5920 - val_loss: 1.1729
Epoch 4/10
1563/1563             9s 5ms/step -
accuracy: 0.6280 - loss: 1.0650 - val_accuracy: 0.6075 - val_loss: 1.1336
Epoch 5/10
1563/1563             9s 5ms/step -
accuracy: 0.6633 - loss: 0.9663 - val_accuracy: 0.6175 - val_loss: 1.1338
Epoch 6/10
1563/1563             9s 5ms/step -
accuracy: 0.6887 - loss: 0.8923 - val_accuracy: 0.6420 - val_loss: 1.0531
Epoch 7/10
1563/1563             9s 5ms/step -
accuracy: 0.7144 - loss: 0.8226 - val_accuracy: 0.6428 - val_loss: 1.0767
Epoch 8/10
1563/1563             9s 5ms/step -
accuracy: 0.7347 - loss: 0.7623 - val_accuracy: 0.6311 - val_loss: 1.1471
Epoch 9/10
1563/1563             9s 5ms/step -
accuracy: 0.7479 - loss: 0.7250 - val_accuracy: 0.6513 - val_loss: 1.1107
Epoch 10/10
1563/1563             9s 5ms/step -
accuracy: 0.7645 - loss: 0.6739 - val_accuracy: 0.6505 - val_loss: 1.1308
313/313             1s 3ms/step -
```

```
accuracy: 0.6548 - loss: 1.1169
CNN test accuracy: 0.6504999995231628
```

[11]:
```
#7. Write a python program to classify different handwritten character images␣
 ↪using the SVM classifier.
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
digits = datasets.load_digits()
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target,␣
 ↪test_size=0.3, random_state=42)
svm = SVC(kernel='linear')
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
print("SVM accuracy on handwritten characters:", accuracy_score(y_test, y_pred))
```

```
SVM accuracy on handwritten characters: 0.9796296296296296
```

[12]:
```
#8. Write a python program to classify different face images using CNN.
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import cifar10
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))
print("CNN test accuracy on face images:", model.evaluate(X_test, y_test)[1])
```

```
C:\Users\User\anaconda3\Lib\site-
packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Epoch 1/10
1563/1563             11s 6ms/step -
```

```
accuracy: 0.3319 - loss: 2.7421 - val_accuracy: 0.4727 - val_loss: 1.4546
Epoch 2/10
1563/1563             9s 6ms/step -
accuracy: 0.5375 - loss: 1.3116 - val_accuracy: 0.5970 - val_loss: 1.1556
Epoch 3/10
1563/1563             9s 6ms/step -
accuracy: 0.6047 - loss: 1.1300 - val_accuracy: 0.6144 - val_loss: 1.1115
Epoch 4/10
1563/1563             9s 6ms/step -
accuracy: 0.6429 - loss: 1.0177 - val_accuracy: 0.6450 - val_loss: 1.0390
Epoch 5/10
1563/1563             9s 6ms/step -
accuracy: 0.6768 - loss: 0.9370 - val_accuracy: 0.6287 - val_loss: 1.0788
Epoch 6/10
1563/1563             9s 6ms/step -
accuracy: 0.6995 - loss: 0.8672 - val_accuracy: 0.6615 - val_loss: 1.0151
Epoch 7/10
1563/1563             9s 6ms/step -
accuracy: 0.7228 - loss: 0.7985 - val_accuracy: 0.6593 - val_loss: 1.0357
Epoch 8/10
1563/1563             10s 6ms/step -
accuracy: 0.7409 - loss: 0.7402 - val_accuracy: 0.6601 - val_loss: 1.0445
Epoch 9/10
1563/1563             10s 6ms/step -
accuracy: 0.7571 - loss: 0.6894 - val_accuracy: 0.6705 - val_loss: 1.0200
Epoch 10/10
1563/1563             9s 6ms/step -
accuracy: 0.7692 - loss: 0.6570 - val_accuracy: 0.6657 - val_loss: 1.0538
313/313             1s 3ms/step -
accuracy: 0.6595 - loss: 1.0582
CNN test accuracy on face images: 0.6657000184059143
```

[13]:
```python
#9. 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 import layers, models
model = models.Sequential([
    layers.Conv2D(64, (3, 3), activation='relu', input_shape=(64, 64, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.RepeatVector(10),
    layers.LSTM(64, return_sequences=True),
    layers.TimeDistributed(layers.Dense(1, activation='sigmoid'))
])
```

```
model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
print("Model summary for gait recognition:")
model.summary()
```

```
Model summary for gait recognition:
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 62, 62, 64)	640
max_pooling2d_5 (MaxPooling2D)	(None, 31, 31, 64)	0
conv2d_6 (Conv2D)	(None, 29, 29, 128)	73,856
max_pooling2d_6 (MaxPooling2D)	(None, 14, 14, 128)	0
flatten_2 (Flatten)	(None, 25088)	0
repeat_vector (RepeatVector)	(None, 10, 25088)	0
lstm (LSTM)	(None, 10, 64)	6,439,168
time_distributed (TimeDistributed)	(None, 10, 1)	65

Total params: 6,513,729 (24.85 MB)

Trainable params: 6,513,729 (24.85 MB)

Non-trainable params: 0 (0.00 B)

[14]:
```
#10. 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 import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
vgg16_model = VGG16(weights='imagenet', include_top=False, input_shape=(224,␣
 ↪224, 3))
densenet_model = DenseNet201(weights='imagenet', include_top=False,␣
 ↪input_shape=(224, 224, 3))
def create_model(base_model):
    model = models.Sequential([
        base_model,
        layers.Flatten(),
        layers.Dense(256, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
    return model
vgg16_cancer_model = create_model(vgg16_model)
densenet_cancer_model = create_model(densenet_model)
print("VGG-16 model summary:")
vgg16_cancer_model.summary()
print("DenseNet-201 model summary:")
densenet_cancer_model.summary()
```

```
VGG-16 model summary:
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              ␣
↪Param #
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         ␣
↪14,714,688
_________________________________________________________________
flatten_3 (Flatten)          (None, 25088)             ␣
↪  0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               ␣
↪6,422,784
_________________________________________________________________
dropout (Dropout)            (None, 256)               ␣
↪  0
```

```
 dense_6 (Dense)                         (None, 1)                                ␣
↪257


 Total params: 21,137,729 (80.63 MB)

 Trainable params: 21,137,729 (80.63 MB)

 Non-trainable params: 0 (0.00 B)
```

DenseNet-201 model summary:

```
Model: "sequential_4"


 Layer (type)                            Output Shape                             ␣
↪Param #

 densenet201 (Functional)                (None, 7, 7, 1920)                       ␣
↪18,321,984

 flatten_4 (Flatten)                     (None, 94080)                            ␣
↪  0

 dense_7 (Dense)                         (None, 256)                              ␣
↪24,084,736

 dropout_1 (Dropout)                     (None, 256)                              ␣
↪  0

 dense_8 (Dense)                         (None, 1)                                ␣
↪257


 Total params: 42,406,977 (161.77 MB)

 Trainable params: 42,177,921 (160.90 MB)

 Non-trainable params: 229,056 (894.75 KB)
```