1.Create a perceptron with appropriate no. of inputs and outputs. Train it using fixed increment learning algorithm until no change in weights is required. Output the final weights.

```
import numpy as np

# Define the input data (2 features)
X = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]])

# Define the target output data (binary classification)
y = np.array([0, 0, 0, 1])

# Initialize weights and bias
weights = np.random.rand(2)  # Assuming 2 input features
bias = np.random.rand(1)

# Learning rate (fixed increment)
learning_rate = 0.1

# Perceptron learning algorithm training
while True:
    weight_change = np.zeros_like(weights)
    bias_change = 0

    for i in range(len(X)):
        # Forward pass (compute the weighted sum)
        print("input is :", X[i])
        weighted_sum = np.dot(X[i], weights) + bias

        # Compute the predicted output (1 if weighted_sum >= 0, 0 otherwise)
        predicted_output = 1 if weighted_sum >= 0 else 0

        # Compute the error
        error = y[i] - predicted_output

        if error != 0:
            # Update weight and bias if there is an error
            weight_change += learning_rate * error * X[i]
            bias_change += learning_rate * error

    # Update weights and bias after processing all inputs in the epoch
    weights += weight_change
    bias += bias_change
    print("weight change:", weight_change)
    print("bias change:", bias_change)
    print()

    # Check for convergence (no change in weights)
    if np.all(weight_change == 0) and bias_change == 0:
        break

    print("# next iteration")

# Output the final weights
print("Final Weights:", weights)
print("Final Bias:", bias)
```

```
input is : [0 0]
input is : [0 1]
input is : [1 0]
input is : [1 1]
weight change: [-0.1 -0.1]
bias change: -0.30000000000000004

# next iteration
input is : [0 0]
input is : [0 1]
input is : [1 0]
input is : [1 1]
weight change: [-0.1 -0.1]
bias change: -0.2

# next iteration
```

```
input is : [0 0]
input is : [0 1]
input is : [1 0]
input is : [1 1]
weight change: [-0.1 -0.1]
bias change: -0.2

# next iteration
input is : [0 0]
input is : [0 1]
input is : [1 0]
input is : [1 1]
weight change: [-0.1  0. ]
bias change: -0.1

# next iteration
input is : [0 0]
input is : [0 1]
input is : [1 0]
input is : [1 1]
weight change: [0. 0.]
bias change: 0

Final Weights: [0.56713484 0.46268719]
Final Bias: [-0.60283124]
```

2.Create a simple ADALINE network with appropriate no. of input and output nodes. Train, it using delta learning rule until no change in weights is required. Output the final weights .

```
import numpy as np

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

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

weights = np.random.rand(2)  # Assuming 2 input features
bias = np.random.rand(1)

learning_rate = 0.1

# Delta learning rule training
prev_weights = weights.copy()  # Store the initial weights for comparison

while True:
    for i in range(len(X)):
        # Forward pass (compute the weighted sum)
        #print('iteration',i)
        #rint("input is:",X[i])
        weighted_sum = np.dot(X[i], weights) + bias

        # Compute the error
        error = y[i] - weighted_sum

        # Update weights and bias

        weights += learning_rate * error * X[i]
        bias += learning_rate * error
        #print("weight change :",weights)
        #print("bias_change :",bias)

    # Check for convergence (small change in weights)
    if np.allclose(weights, prev_weights):
        break
    else:
        prev_weights = weights.copy()

# Output the final weights
print("Final Weights:", weights)
print("Final Bias:", bias)
```

```
Final Weights: [0.55548949 0.52770749]
Final Bias: [-0.27768576]
```

3 Train the autocorrelator by given patterns: AI=(-1,1,-1,1), A2=(1,1,1,-1), A3=(-1,-1,-1,1). Test it using patterns: Ax=(-1,1,-1,1), Ay=(1,1,1,1), Az=(-1,-1,-1,-1).

Double-click (or enter) to edit

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Define training patterns
X_train = np.array([[-1, 1, -1, 1],
                    [1, 1, 1, -1],
                    [-1, -1, -1, 1]])

# Use the same patterns as target output for an autocorrelator
y_train = X_train

# Build a simple feedforward neural network
model = Sequential()
model.add(Dense(units=8, input_dim=4, activation='relu'))
model.add(Dense(units=4, activation='linear'))  # Output layer with linear activation

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model.fit(X_train, y_train, epochs=1000, verbose=0)

# Test the autocorrelator with given test patterns
X_test = np.array([[-1, 1, -1, 1],
                   [1, 1, 1, 1],
                   [-1, -1, -1, -1]])

# Predict the output for test patterns
predictions = model.predict(X_test)

# Output the predictions
for i, pattern in enumerate(X_test):
    print(f"Pattern {i + 1}: {pattern} => Prediction: {predictions[i]}")
```

```
# Assuming you want to implement an ADALINE class:

class Adaline:
    def __init__(self, input_size, learning_rate=0.01):
        self.weights = np.random.rand(input_size)
        self.bias = np.random.rand(1)
        self.learning_rate = learning_rate

    def predict(self, X):
        return np.dot(X, self.weights) + self.bias

    def train(self, X, y, epochs=1):
        for _ in range(epochs):
            for i in range(len(X)):
                prediction = self.predict(X[i])
                error = y[i] - prediction
                # Update weights and bias using the Adaline learning rule
                self.weights += self.learning_rate * error * X[i]
                self.bias += self.learning_rate * error

# Initialize your ADALINE model
A1 = np.array([-1, 1, -1, 1])
A2 = np.array([1, 1, 1, -1])
A3 = np.array([-1, -1, -1, 1])

# Define the target outputs for A1, A2, A3
target_A1 = 1
target_A2 = 1
target_A3 = 1

# Initialize Adaline
input_size = len(A1)
```

```
adaline = Adaline(input_size)

# Train the network with the given patterns
adaline.train(np.array([A1, A2, A3]), np.array([target_A1, target_A2, target_A3]), epochs=100)

# Test the network with the test patterns
Ax = np.array([1, -1, 1, -1])
Ay = np.array([-1, -1, 1, 1])
Az = np.array([1, 1, -1, -1])

output_Ax = adaline.predict(Ax)
output_Ay = adaline.predict(Ay)
output_Az = adaline.predict(Az)

# Output the results
print("Output for Ax:", output_Ax)
print("Output for Ay:", output_Ay)
print("Output for Az:", output_Az)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
Pattern 1: [-1  1 -1  1] => Prediction: [-0.62343967  0.5031746  -0.9034304   0.9793873 ]
Pattern 2: [1 1 1 1] => Prediction: [ 0.7309717   2.3362682   0.10962461 -0.84006286]
Pattern 3: [-1 -1 -1 -1] => Prediction: [-0.9260362 -0.4228353 -1.0211328  1.3588169]
Output for Ax: [0.68063251]
Output for Ay: [1.92455461]
Output for Az: [-0.15905996]
```

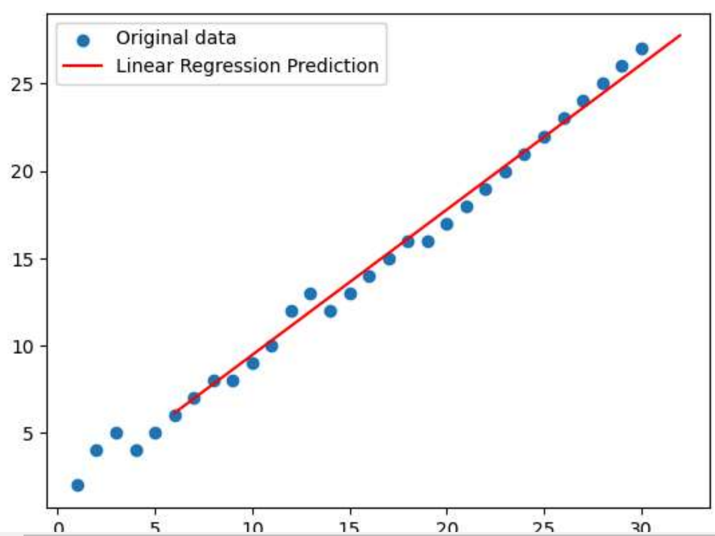
4. Train the hetro correlator using multiple training encoding strategy for given patterns A1-(000111001) B1-(010000111), A2-(111001110) B2-(100000001), A3=(110110101) B3(101001010). Test it using pattern A2

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Define training patterns and their corresponding labels
X_train = np.array([[0, 0, 0, 1, 1, 1, 0, 0, 1],
                    [0, 1, 0, 0, 0, 0, 1, 1, 1],
                    [1, 1, 1, 0, 0, 1, 1, 1, 0],
                    [1, 0, 0, 0, 0, 0, 0, 0, 1]])

# Corresponding labels: 0 for A, 1 for B
y_train = np.array([0, 1, 0, 1])

# Build a simple feedforward neural network
model = Sequential()
model.add(Dense(units=8, input_dim=9, activation='relu'))  # Hidden layer with 8 units
model.add(Dense(units=1, activation='sigmoid'))  # Output layer with sigmoid activation

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=1000, verbose=0)  # Suppress verbosity for training

# Test the hetero-correlator with a given test pattern A2
X_test = np.array([[1, 1, 1, 0, 0, 1, 1, 1, 0]])

# Predict the output for the test pattern
prediction = model.predict(X_test)

# Output the prediction
print(f"Test Pattern A2: {X_test.flatten()} => Prediction: {prediction[0][0]}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
Test Pattern A2: [1 1 1 0 0 1 1 1 0] => Prediction: 0.01918846368789673
```

5 Implement Linear/Logistic regression ?

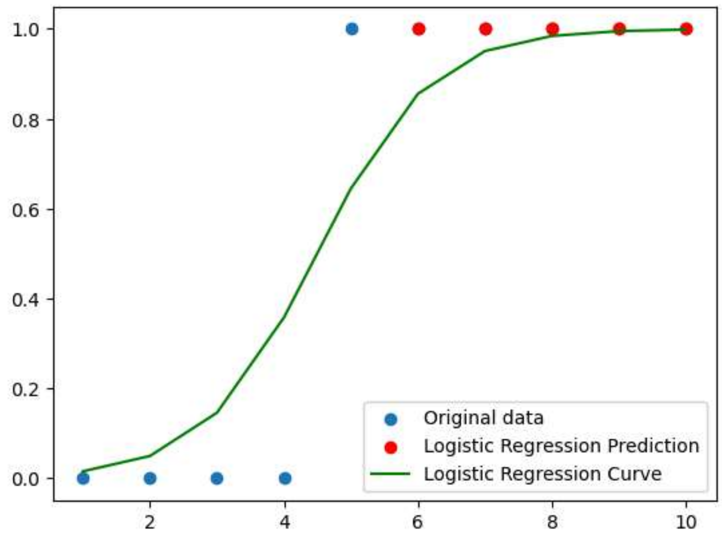
```
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Generate more sample data (3 times the original)
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
              11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
              21, 22, 23, 24, 25, 26, 27, 28, 29, 30]).reshape(-1, 1)
y = np.array([2, 4, 5, 4, 5, 6, 7, 8, 8, 9,
              10, 12, 13, 12, 13, 14, 15, 16, 16, 17,
              18, 19, 20, 21, 22, 23, 24, 25, 26, 27])

# Create a linear regression model
linear_reg = LinearRegression()

# Train the model
linear_reg.fit(X, y)

# Make predictions
X_pred = np.array([6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
                   16, 17, 18, 19, 20, 21, 22, 23, 24, 25,
                   26, 27, 28, 29, 30, 31, 32]).reshape(-1, 1)
y_pred = linear_reg.predict(X_pred)

# Plot the results
plt.scatter(X, y, label='Original data')
plt.plot(X_pred, y_pred, label='Linear Regression Prediction', color='red')
plt.legend()
plt.show()
```



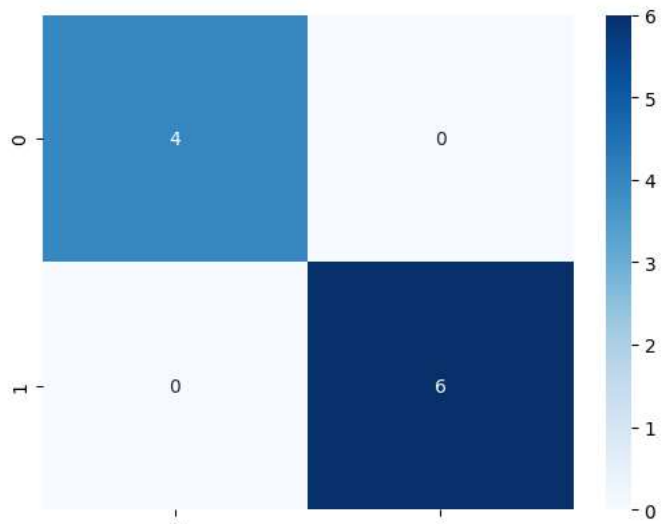
Logistic Regression:

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Generate some sample data
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]).reshape(-1, 1)
y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

# Create a logistic regression model
logistic_reg = LogisticRegression()

# Train the model
logistic_reg.fit(X, y)

# Make predictions on the original data and new data
X_pred = np.array([6, 7, 8, 9, 10]).reshape(-1, 1)
y_pred = logistic_reg.predict(X_pred)
```

```
# Calculate accuracy
accuracy = accuracy_score(y, logistic_reg.predict(X))

# Generate confusion matrix
conf_matrix = confusion_matrix(y, logistic_reg.predict(X))

# Plot the results
plt.scatter(X, y, label='Original data')
plt.scatter(X_pred, y_pred, label='Logistic Regression Prediction', color='red')
plt.plot(X, logistic_reg.predict_proba(X)[:, 1], label='Logistic Regression Curve', color='green')
plt.legend()
plt.show()

# Display accuracy and confusion matrix
print(f'Accuracy: {accuracy}')
print('Confusion Matrix:')
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g')
plt.show()
```




```
Accuracy: 1.0
Confusion Matrix:
```



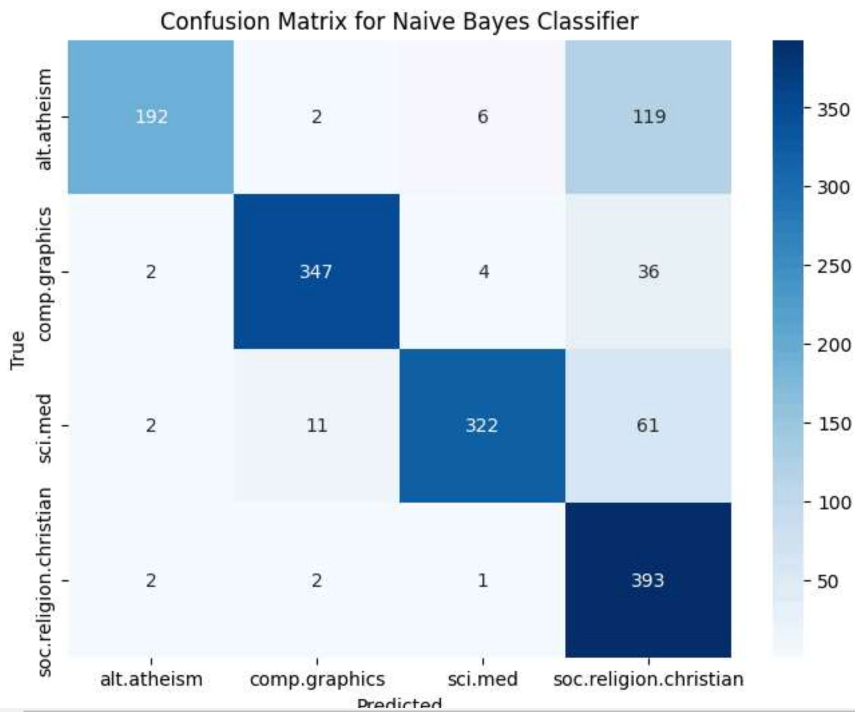
6 Implementation of Naive Bayes/SVM/SGD/SVM classifier on text and image

1. Naive Bayes Classifier:

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Load the 20 Newsgroups dataset
categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.med']
newsgroups_train = fetch_20newsgroups(subset='train', categories=categories)
newsgroups_test = fetch_20newsgroups(subset='test', categories=categories)

# Convert text data to TF-IDF vectors
vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(newsgroups_train.data)
X_test = vectorizer.transform(newsgroups_test.data)

# Create Naive Bayes classifier
nb_classifier = MultinomialNB()

# Train the classifier
nb_classifier.fit(X_train, newsgroups_train.target)

# Make predictions
nb_predictions = nb_classifier.predict(X_test)

# Evaluate the classifier
print("Naive Bayes Classifier:")
print("Accuracy:", accuracy_score(newsgroups_test.target, nb_predictions))
print("Classification Report:\n", classification_report(newsgroups_test.target, nb_predictions))

# Optionally, plot the confusion matrix
conf_matrix = confusion_matrix(newsgroups_test.target, nb_predictions)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=newsgroups_train.target_names, yticklabels=newsgroups_train.target_na
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for Naive Bayes Classifier')
plt.show()
```

```
Naive Bayes Classifier:
Accuracy: 0.8348868175765646
Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.60      0.74       319
           1       0.96      0.89      0.92       389
           2       0.97      0.81      0.88       396
           3       0.65      0.99      0.78       398

    accuracy                           0.83      1502
   macro avg       0.89      0.82      0.83      1502
weighted avg       0.88      0.83      0.84      1502
```



2. SVM Classifier:

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Create SVM classifier with linear kernel
svm_classifier = SVC(kernel='linear')

# Train the classifier
svm_classifier.fit(X_train, newsgroups_train.target)

# Make predictions
svm_predictions = svm_classifier.predict(X_test)

# Evaluate the classifier
print("\nSVM Classifier:")
print("Accuracy:", accuracy_score(newsgroups_test.target, svm_predictions))
print("Classification Report:\n", classification_report(newsgroups_test.target, svm_predictions))

# Optionally, plot the confusion matrix
conf_matrix = confusion_matrix(newsgroups_test.target, svm_predictions)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=newsgroups_train.target_names, yticklabels=newsgroups_train.target_na
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for SVM Classifier')
plt.show()
```

```
SVM Classifier:
Accuracy: 0.9207723035952063
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.83      0.89       319
           1       0.90      0.96      0.93       389
           2       0.94      0.91      0.93       396
           3       0.89      0.96      0.93       398

    accuracy                           0.92      1502
   macro avg       0.93      0.92      0.92      1502
weighted avg       0.92      0.92      0.92      1502
```

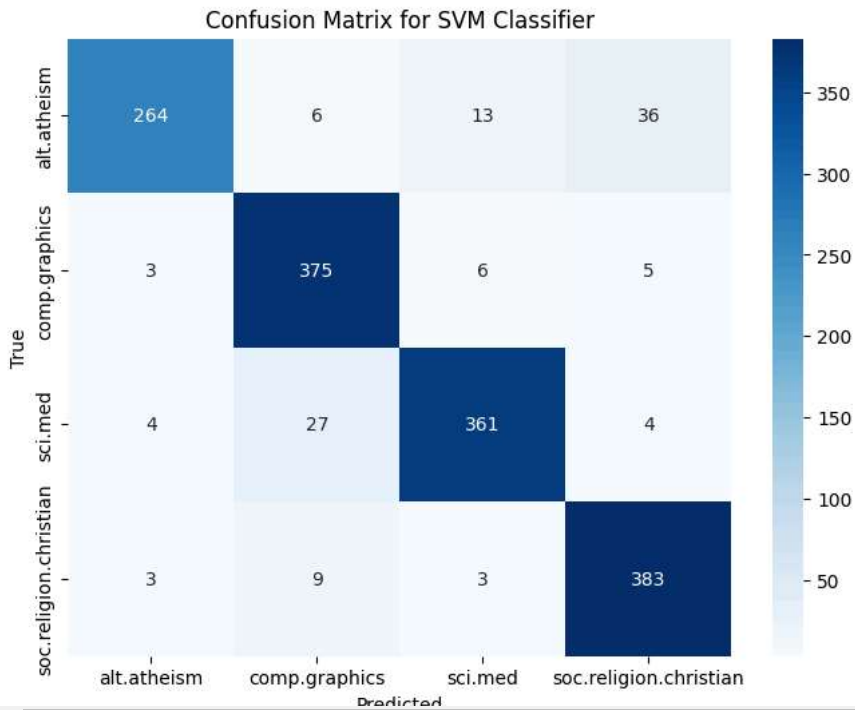


Image Classification:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Load the digits dataset
digits = datasets.load_digits()

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target, test_size=0.2, random_state=42)

# Scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create SVM classifier
svm_classifier_digits = SVC()

# Train the classifier
svm_classifier_digits.fit(X_train, y_train)

# Make predictions
svm_predictions_digits = svm_classifier_digits.predict(X_test)

# Evaluate the classifier
print("SVM Classifier (Digits Dataset):")
print("Accuracy:", accuracy_score(y_test, svm_predictions_digits))
```

```
print("Classification Report:\n", classification_report(y_test, svm_predictions_digits))
```

```
SVM Classifier (Digits Dataset):
Accuracy: 0.9805555555555555
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        33
           1       1.00      1.00      1.00        28
           2       1.00      1.00      1.00        33
           3       1.00      0.97      0.99        34
           4       0.96      1.00      0.98        46
           5       0.96      0.98      0.97        47
           6       0.97      1.00      0.99        35
           7       1.00      0.94      0.97        34
           8       0.97      0.97      0.97        30
           9       0.97      0.95      0.96        40

    accuracy                           0.98       360
   macro avg       0.98      0.98      0.98       360
weighted avg       0.98      0.98      0.98       360
```

2. SGD Classifier:

```
from sklearn.linear_model import SGDClassifier

# Create SGD classifier
sgd_classifier_digits = SGDClassifier()

# Train the classifier
sgd_classifier_digits.fit(X_train, y_train)

# Make predictions
sgd_predictions_digits = sgd_classifier_digits.predict(X_test)

# Evaluate the classifier
print("\nSGD Classifier (Digits Dataset):")
print("Accuracy:", accuracy_score(y_test, sgd_predictions_digits))
print("Classification Report:\n", classification_report(y_test, sgd_predictions_digits))
```

```

SGD Classifier (Digits Dataset):
Accuracy: 0.9527777777777777
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        33
           1       0.93      0.93      0.93        28
           2       1.00      0.97      0.98        33
           3       1.00      0.97      0.99        34
           4       1.00      0.98      0.99        46
           5       0.94      0.94      0.94        47
           6       1.00      0.97      0.99        35
           7       1.00      0.97      0.99        34
           8       0.78      0.93      0.85        30
           9       0.90      0.88      0.89        40

    accuracy                           0.95       360
   macro avg       0.95      0.95      0.95       360
weighted avg       0.96      0.95      0.95       360
```

Image Classification using CNN:

```
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import datasets

# Load the digits dataset
digits = datasets.load_digits()
```

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target, test_size=0.2, random_state=42)

# Reshape data to fit a 4D tensor (image format)
X_train = X_train.reshape(X_train.shape[0], 8, 8, 1)
X_test = X_test.reshape(X_test.shape[0], 8, 8, 1)

# Normalize pixel values to be between 0 and 1
X_train, X_test = X_train / 255.0, X_test / 255.0

# Create a CNN model
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(8, 8, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(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)
print(f"\nTest Accuracy: {test_acc}")
```

```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
45/45 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.1609 - loss: 2.2969 - val_accuracy: 0.1833 - val_loss: 2.2738
Epoch 2/10
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.2182 - loss: 2.2507 - val_accuracy: 0.4083 - val_loss: 2.1680
Epoch 3/10
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.4751 - loss: 2.1175 - val_accuracy: 0.5528 - val_loss: 1.9392
Epoch 4/10
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5737 - loss: 1.8513 - val_accuracy: 0.7056 - val_loss: 1.5681
Epoch 5/10
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7493 - loss: 1.4733 - val_accuracy: 0.7917 - val_loss: 1.1946
Epoch 6/10
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7794 - loss: 1.1143 - val_accuracy: 0.8361 - val_loss: 0.8896
Epoch 7/10
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8271 - loss: 0.8604 - val_accuracy: 0.8778 - val_loss: 0.7140
Epoch 8/10
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8522 - loss: 0.7182 - val_accuracy: 0.8861 - val_loss: 0.6017
Epoch 9/10
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8839 - loss: 0.5847 - val_accuracy: 0.8972 - val_loss: 0.5233
Epoch 10/10
45/45 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8932 - loss: 0.5219 - val_accuracy: 0.9083 - val_loss: 0.4670
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9159 - loss: 0.4487

Test Accuracy: 0.9083333611488342
```

Text Classification using CNN:

```
import tensorflow as tf

from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.datasets import fetch_20newsgroups

# Load the 20 Newsgroups dataset
categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.med']
newsgroups_train = fetch_20newsgroups(subset='train', categories=categories)
newsgroups_test = fetch_20newsgroups(subset='test', categories=categories)

# Convert text data to TF-IDF vectors
vectorizer = TfidfVectorizer(max_features=5000)
X_train = vectorizer.fit_transform(newsgroups_train.data).toarray()
X_test = vectorizer.transform(newsgroups_test.data).toarray()

# Reshape data to fit a 3D tensor (text data format)
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
```

```
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

# Create a CNN model for text classification
model_text = models.Sequential()
model_text.add(layers.Conv1D(128, 5, activation='relu', input_shape=(X_train.shape[1], 1)))
model_text.add(layers.GlobalMaxPooling1D())
model_text.add(layers.Dense(64, activation='relu'))
model_text.add(layers.Dense(4, activation='softmax'))

# Compile the model
model_text.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model
model_text.fit(X_train, newsgroups_train.target, epochs=5, validation_data=(X_test, newsgroups_test.target))

# Evaluate the model
test_loss, test_acc = model_text.evaluate(X_test, newsgroups_test.target)
print(f"\nTest Accuracy: {test_acc}")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/5
71/71 ━━━━━━━━━━━━━━━━━━━━ 25s 332ms/step - accuracy: 0.2924 - loss: 1.3805 - val_accuracy: 0.2730 - val_loss: 1.3733
Epoch 2/5
71/71 ━━━━━━━━━━━━━━━━━━━━ 42s 343ms/step - accuracy: 0.3375 - loss: 1.3615 - val_accuracy: 0.3502 - val_loss: 1.3585
Epoch 3/5
71/71 ━━━━━━━━━━━━━━━━━━━━ 43s 366ms/step - accuracy: 0.3438 - loss: 1.3516 - val_accuracy: 0.3216 - val_loss: 1.3434
Epoch 4/5
71/71 ━━━━━━━━━━━━━━━━━━━━ 47s 455ms/step - accuracy: 0.3358 - loss: 1.3340 - val_accuracy: 0.3329 - val_loss: 1.3377
Epoch 5/5
71/71 ━━━━━━━━━━━━━━━━━━━━ 32s 333ms/step - accuracy: 0.3626 - loss: 1.3208 - val_accuracy: 0.3515 - val_loss: 1.3356
47/47 ━━━━━━━━━━━━━━━━━━━━ 6s 134ms/step - accuracy: 0.3479 - loss: 1.3427

Test Accuracy: 0.3515312969684601
```

8 To study Word Embedding techniques: Word2vec, doc2vec, Glove

```
import numpy as np
from gensim.models import Word2Vec, Doc2Vec
from gensim.models.doc2vec import TaggedDocument
from nltk.tokenize import word_tokenize
import spacy

# Download NLTK punkt tokenizer
import nltk
nltk.download('punkt')  # This downloads the 'punkt' tokenizer
nltk.download('punkt_tab')  # If 'punkt' doesn't resolve the issue, download 'punkt_tab'

# Sample sentences for Word2Vec
sentences = [
    "Word embeddings provide a dense representation of words.",
    "They capture semantic relationships and context in language.",
    "Word2Vec is a popular word embedding technique."
]

# Tokenize sentences
tokenized_sentences = [word_tokenize(sentence.lower()) for sentence in sentences]

# Train Word2Vec model
model_w2v = Word2Vec(sentences=tokenized_sentences, vector_size=100, window=5, sg=1, min_count=1)

# Get the vector representation of a word
word_vector = model_w2v.wv['word']
print("Word Vector for 'word' (Word2Vec):", word_vector)

# Sample documents for Doc2Vec
documents = [
    "Word embeddings provide a dense representation of words.",
    "They capture semantic relationships and context in language.",
    "Doc2Vec extends Word2Vec to learn document representations."
]

# Tokenize and tag documents
tagged_data = [TaggedDocument(words=word_tokenize(doc.lower()), tags=[str(i)]) for i, doc in enumerate(documents)]

# Train Doc2Vec model
```

```
# Train Doc2Vec model
model_d2v = Doc2Vec(vector_size=100, window=5, min_count=1, workers=4, epochs=100)
model_d2v.build_vocab(tagged_data)
model_d2v.train(tagged_data, total_examples=model_d2v.corpus_count, epochs=model_d2v.epochs)

# Get the vector representation of a document
doc_vector = model_d2v.dv['0']  # Accessing document vector using 'dv'
print("Document Vector for Document 0 (Doc2Vec):", doc_vector)

# Load spaCy with GloVe pre-trained embeddings (after installing the model)
nlp = spacy.load("en_core_web_md")

# Get the vector representation of a word (GloVe)
word_vector_glove = nlp("word").vector
print("Word Vector for 'word' (GloVe):", word_vector_glove)

# Get the vector representation of a document (GloVe)
doc_vector_glove = nlp("Word embeddings provide a dense representation of words.").vector
print("Document Vector (GloVe):", doc_vector_glove)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
Word Vector for 'word' (Word2Vec): [-8.6202472e-03  3.6648309e-03  5.1877792e-03  5.7448559e-03
  7.4660280e-03 -6.1680586e-03  1.1061627e-03  6.0497751e-03
 -2.8410261e-03 -6.1753229e-03 -4.0880564e-04 -8.3705420e-03
 -5.6011369e-03  7.1056709e-03  3.3504120e-03  7.2252913e-03
  6.8017221e-03  7.5302417e-03 -3.7888847e-03 -5.6657044e-04
  2.3481909e-03 -4.5174705e-03  8.3888695e-03 -9.8560909e-03
  6.7665288e-03  2.9130057e-03 -4.9344390e-03  4.3987152e-03
 -1.7417739e-03  6.7099077e-03  9.9680964e-03 -4.3630879e-03
 -5.9757102e-04 -5.6949523e-03  3.8469101e-03  2.7875938e-03
  6.8915561e-03  6.1035752e-03  9.5393378e-03  9.2725139e-03
  7.9010539e-03 -6.9900132e-03 -9.1585424e-03 -3.5591828e-04
 -3.1001300e-03  7.8935390e-03  5.9350259e-03 -1.5454330e-03
  1.5119562e-03  1.7927517e-03  7.8198109e-03 -9.5110545e-03
 -2.0604000e-04  3.4711380e-03 -9.4124721e-04  8.3783632e-03
  9.0116607e-03  6.5322369e-03 -7.1438210e-04  7.7077537e-03
 -8.5349148e-03  3.2087313e-03 -4.6323333e-03 -5.0909757e-03
  3.5859107e-03  5.3728130e-03  7.7685528e-03 -5.7690130e-03
  7.4317581e-03  6.6229636e-03 -3.7085600e-03 -8.7417038e-03
  5.4361718e-03  6.5079029e-03 -7.8477210e-04 -6.7096367e-03
 -7.0837936e-03 -2.4956004e-03  5.1437700e-03 -3.6645704e-03
 -9.3697058e-03  3.8237025e-03  4.8841462e-03 -6.4278888e-03
  1.2067747e-03 -2.0763762e-03  2.5281282e-05 -9.8839961e-03
  2.6907038e-03 -4.7467933e-03  1.0862816e-03 -1.5726023e-03
  2.1978270e-03 -7.8789806e-03 -2.7119482e-03  2.6614545e-03
  5.3461893e-03 -2.3909810e-03 -9.5094284e-03  4.5079072e-03]
Document Vector for Document 0 (Doc2Vec): [-0.00865385 -0.00868342 -0.01115877  0.01116269  0.0044116  -0.00131533
 -0.01010796 -0.00458529 -0.01133068  0.00240048  0.00233414  0.0046878
 -0.0065164  -0.00480197 -0.00130254 -0.01039608  0.00140149  0.0105676
 -0.01097117 -0.00516461 -0.00392581  0.00341202 -0.00611344  0.00559098
  0.00467996 -0.00865612 -0.01224211 -0.01119974  0.00512839 -0.01182436
```