In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

In [ ]:
```
from sklearn import datasets
```

In [ ]:
```
digits = datasets.load_digits()
```

In [ ]:
```
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))
labels = digits.target
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)
```

Logistic regressor:

In [ ]:
```
model = LogisticRegression(C=0.1, solver="saga", max_iter=100)
```

In [ ]:
```
model.fit(X_train, y_train)
```

```
c:\Users\Vitrag Khatadia\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\linear_model\_sag.py:350: ConvergenceWarning: The max_i
ter was reached which means the coef_ did not converge
  warnings.warn(
```

Out[ ]:
```
▾ LogisticRegression
LogisticRegression(C=0.1, solver='saga')
```

In [ ]:
```
y_pred = model.predict(X_test)
```

In [ ]:
```
from sklearn.metrics import accuracy_score, classification_report
```

In [ ]:
```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.97
```

In [ ]:
```
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        33
           1       0.97      1.00      0.98        28
           2       0.97      1.00      0.99        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       0.97      0.97      0.97        35
           7       1.00      0.97      0.99        34
           8       0.97      0.97      0.97        30
           9       0.95      0.97      0.96        40

    accuracy                           0.97       360
   macro avg       0.98      0.98      0.98       360
weighted avg       0.98      0.97      0.98       360
```

SVM:

In [ ]:
```
from sklearn import svm
```

In [ ]:
```
model = svm.SVC(C=10,kernel='rbf',gamma=0.001)
```

In [ ]:
```
model.fit(X_train, y_train)
```

Out[ ]:
```
▾ SVC
SVC(C=10, gamma=0.001)
```

In [ ]:
```
y_pred = model.predict(X_test)
```

In [ ]:
```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.99
```

In [ ]:
```
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
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       1.00      1.00      1.00        46
           5       0.98      0.98      0.98        47
           6       0.97      1.00      0.99        35
           7       0.97      0.97      0.97        34
           8       1.00      1.00      1.00        30
           9       0.97      0.97      0.97        40

    accuracy                           0.99       360
   macro avg       0.99      0.99      0.99       360
weighted avg       0.99      0.99      0.99       360
```

Decision tree:

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import fetch_openml

# Load the MNIST dataset
mnist = fetch_openml('mnist_784', version=1)

# Split the dataset into features and labels
X = mnist.data
y = mnist.target

# Define the decision tree classifier
model = DecisionTreeClassifier()

# Define the hyperparameters to be tuned
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Perform grid search to find the best hyperparameters
grid_search = GridSearchCV(model, param_grid, cv=5)
grid_search.fit(X, y)
```

```
# Print the best hyperparameters
print("Best Hyperparameters:")
print(grid_search.best_params_)
```

In [ ]:
```
model = DecisionTreeClassifier()
```

In [ ]:
```
model.fit(X_train, y_train)
```

Out[ ]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [ ]:
```
y_pred = model.predict(X_test)
```

In [ ]:
```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 0.85
```

In [ ]:
```
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.88      0.92        33
           1       0.96      0.79      0.86        28
           2       0.90      0.79      0.84        33
           3       0.67      0.91      0.78        34
           4       0.76      0.89      0.82        46
           5       0.95      0.83      0.89        47
           6       0.91      0.91      0.91        35
           7       0.88      0.88      0.88        34
           8       0.79      0.73      0.76        30
           9       0.82      0.82      0.82        40

    accuracy                           0.85       360
   macro avg       0.86      0.84      0.85       360
weighted avg       0.86      0.85      0.85       360
```

CNN:

In [ ]:
```
# Import necessary libraries
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
```

```
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.metrics import accuracy_score, classification_report

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


data = digits.images / 16.0
labels = digits.target

X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)

X_train = X_train.reshape(X_train.shape[0], 8, 8, 1)
X_test = X_test.reshape(X_test.shape[0], 8, 8, 1)

# Convert labels to one-hot encoding
y_train = to_categorical(y_train, num_classes=10)      # Convert training labels to one-hot encoding
y_test = to_categorical(y_test, num_classes=10)        # Convert testing labels to one-hot encoding

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(8, 8, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))


accuracy = model.evaluate(X_test, y_test)[1]
print(f"Accuracy: {accuracy:.2f}")
```

```
WARNING:tensorflow:From c:\Users\Vitrag Khatadia\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\backend.py:873: The name tf.g
et_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From c:\Users\Vitrag Khatadia\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\layers\pooling\max_pooling2d.
py:161: The name tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.

WARNING:tensorflow:From c:\Users\Vitrag Khatadia\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\optimizers\__init__.py:309: T
he name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

Epoch 1/10
WARNING:tensorflow:From c:\Users\Vitrag Khatadia\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\utils\tf_utils.py:492: The na
me tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From c:\Users\Vitrag Khatadia\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\engine\base_layer_utils.py:38
4: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

45/45 [==============================] - 1s 9ms/step - loss: 2.1135 - accuracy: 0.4043 - val_loss: 1.8146 - val_accuracy: 0.7306
Epoch 2/10
45/45 [==============================] - 0s 4ms/step - loss: 1.3564 - accuracy: 0.8267 - val_loss: 0.9186 - val_accuracy: 0.8000
Epoch 3/10
45/45 [==============================] - 0s 6ms/step - loss: 0.6400 - accuracy: 0.8998 - val_loss: 0.4646 - val_accuracy: 0.9111
Epoch 4/10
45/45 [==============================] - 0s 8ms/step - loss: 0.3810 - accuracy: 0.9207 - val_loss: 0.2894 - val_accuracy: 0.9389
Epoch 5/10
45/45 [==============================] - 0s 6ms/step - loss: 0.2678 - accuracy: 0.9443 - val_loss: 0.2207 - val_accuracy: 0.9472
Epoch 6/10
45/45 [==============================] - 0s 6ms/step - loss: 0.2126 - accuracy: 0.9513 - val_loss: 0.1784 - val_accuracy: 0.9528
Epoch 7/10
45/45 [==============================] - 0s 4ms/step - loss: 0.1772 - accuracy: 0.9589 - val_loss: 0.1503 - val_accuracy: 0.9694
Epoch 8/10
45/45 [==============================] - 0s 7ms/step - loss: 0.1514 - accuracy: 0.9652 - val_loss: 0.1448 - val_accuracy: 0.9667
Epoch 9/10
45/45 [==============================] - 0s 7ms/step - loss: 0.1349 - accuracy: 0.9694 - val_loss: 0.1227 - val_accuracy: 0.9667
Epoch 10/10
45/45 [==============================] - 0s 7ms/step - loss: 0.1207 - accuracy: 0.9722 - val_loss: 0.1137 - val_accuracy: 0.9722
12/12 [==============================] - 0s 2ms/step - loss: 0.1137 - accuracy: 0.9722
Accuracy: 0.97
```

```
# Import necessary libraries
from tensorflow.keras import Sequential        # Import the Sequential model from Keras
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense   # Import layers for building the CNN
from tensorflow.keras.utils import to_categorical   # Import utility functions
from sklearn.model_selection import train_test_split # Import train_test_split to split data
from sklearn import datasets                      # Import datasets module from scikit-learn
from sklearn.metrics import accuracy_score, classification_report # Import metrics for evaluation

# Load the MNIST dataset
digits = datasets.load_digits()   # Load the digits dataset from scikit-learn

# Normalize pixel values to be between 0 and 1
data = digits.images / 16.0         # Normalize pixel values to be between 0 and 1
labels = digits.target              # Get the target labels

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)
# Split the dataset into training and testing sets using 80% for training and 20% for testing

# Reshape the data for compatibility with CNN
X_train = X_train.reshape(X_train.shape[0], 8, 8, 1)   # Reshape training data to include a single channel
X_test = X_test.reshape(X_test.shape[0], 8, 8, 1)      # Reshape testing data to include a single channel

# Convert labels to one-hot encoding
y_train = to_categorical(y_train, num_classes=10)      # Convert training labels to one-hot encoding
y_test = to_categorical(y_test, num_classes=10)        # Convert testing labels to one-hot encoding

# Create a CNN model
model = Sequential()                 # Create a Sequential model
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(8, 8, 1)))  # Add a convolutional layer
model.add(MaxPooling2D((2, 2)))      # Add a max pooling layer
model.add(Flatten())                 # Flatten the output for the fully connected layers
model.add(Dense(64, activation='relu'))  # Add a fully connected layer with ReLU activation
model.add(Dense(10, activation='softmax')) # Add the output layer with softmax activation for multiclass classification

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

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))
# Train the CNN model on the training data for 10 epochs with a batch size of 32

# Evaluate the model
accuracy = model.evaluate(X_test, y_test)[1]  # Evaluate the accuracy on the test set
print(f"Accuracy: {accuracy:.2f}")          # Print the accuracy of the model
```

```
Epoch 1/10
45/45 [==============================] - 1s 9ms/step - loss: 2.0798 - accuracy: 0.5024 - val_loss: 1.7711 - val_accuracy: 0.7972
Epoch 2/10
45/45 [==============================] - 0s 4ms/step - loss: 1.3092 - accuracy: 0.8302 - val_loss: 0.8814 - val_accuracy: 0.8250
Epoch 3/10
45/45 [==============================] - 0s 4ms/step - loss: 0.6120 - accuracy: 0.8907 - val_loss: 0.4520 - val_accuracy: 0.9056
Epoch 4/10
45/45 [==============================] - 0s 4ms/step - loss: 0.3564 - accuracy: 0.9241 - val_loss: 0.2926 - val_accuracy: 0.9222
Epoch 5/10
```

```
45/45 [==============================] - 0s 4ms/step - loss: 0.2471 - accuracy: 0.9513 - val_loss: 0.2213 - val_accuracy: 0.9472
Epoch 6/10
45/45 [==============================] - 0s 4ms/step - loss: 0.1932 - accuracy: 0.9576 - val_loss: 0.1769 - val_accuracy: 0.9583
Epoch 7/10
45/45 [==============================] - 0s 4ms/step - loss: 0.1588 - accuracy: 0.9659 - val_loss: 0.1542 - val_accuracy: 0.9639
Epoch 8/10
45/45 [==============================] - 0s 3ms/step - loss: 0.1326 - accuracy: 0.9749 - val_loss: 0.1586 - val_accuracy: 0.9611
Epoch 9/10
45/45 [==============================] - 0s 4ms/step - loss: 0.1234 - accuracy: 0.9715 - val_loss: 0.1154 - val_accuracy: 0.9667
Epoch 10/10
45/45 [==============================] - 0s 4ms/step - loss: 0.1020 - accuracy: 0.9770 - val_loss: 0.1074 - val_accuracy: 0.9750
12/12 [==============================] - 0s 2ms/step - loss: 0.1074 - accuracy: 0.9750
Accuracy: 0.98
```

Observation and Learning

When comparing all models implemented above, support vector machine performed the best with accuracy score of 99% and decision tree performed the worst with accuracy score of 85%.

Logistic regression was not suitable for this problem because in train data even small fluctuation can cause change in output and in logistic it generates a arbitrary line which does not provide accurate classification.

Decision tree was also not suitable for this problem because it will generate branches for small fluctutations in data which can cause overfitting.

SVC was the most suitable model as it was able to provide the best classification line and in turn gave the best accuracy

CNN was also suitable and can provide the best accuracy based on parameter tuning

Conclusion

In conclusion, building a handwritten digit recognition system using machine learning in Python involves selecting the appropriate dataset, preprocessing the data, choosing a suitable model, training the model, evaluating its performance, and developing a user interface for interactive use.