HANDWRITTEN DIGIT RECOGNITION USING SUPPORT VECTOR MACHINES

Submitted By:- Piyush - 2021BEC0023

Introduction:

In this project, we delve into the realm of handwritten digit recognition, a pivotal area within machine learning and computer vision that focuses on accurately identifying numerical values from images of handwritten digits. Central to our exploration is the MNIST dataset, renowned for its extensive collection of 28x28 pixel grayscale images depicting digits 0 to 9. This dataset not only serves as a benchmark for image processing algorithms but also plays a crucial role in advancing research and applications in the field.

Support Vector Machine:

At the heart of our digit recognition system is the Support Vector Machine (SVM) algorithm, celebrated for its proficiency in classification tasks. SVM stands out due to its ability to construct an optimal hyperplane in a high-dimensional space, effectively segregating classes with maximum margin. This characteristic makes SVM particularly well-suited for the challenge of digit recognition, where the goal is to navigate through the complex, high-dimensional data of images to achieve precise classification.

MNIST:

This project aims to showcase the practical application of SVM on the MNIST dataset, emphasizing the comprehensive process from initial data preprocessing and model training to the final evaluation of the model's performance on unseen data. By leveraging SVM's robust capabilities in finding distinctive boundaries within the data, we embark on a detailed journey to unravel the intricacies of implementing an efficient digit recognition system. Through this endeavor, we aim to highlight the versatility of SVM across various sectors, including healthcare, signal processing, and beyond, demonstrating its substantial impact on the field of machine learning.

CODES:

1. Data Collection:

Dataset Used: MNIST dataset, containing 70,000 images of handwritten digits.
Format: The dataset was accessed in a format compatible with Python, specifically using the CSV format to facilitate easy loading and manipulation using pandas or direct fetching through scikit-learn utilities.

```python
# Piyush - 2021BEC0023      You, 19 minutes ago • Uncommitted changes
import numpy as np
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.preprocessing import MinMaxScaler
import pickle


mnist = fetch_openml('mnist_784', version=1)
X, y = mnist["data"], mnist["target"]
```

2. Data Preprocessing:

Process Overview: The dataset was divided into training and testing sets to prepare for model training and evaluation.
Preprocessing Steps:
Normalization: Feature scaling was applied using MinMaxScaler to normalize pixel values to a range of 0 to 1.
Splitting: The dataset was split into training (60,000 samples) and testing (10,000 samples) sets.

```python
# Normalize the data
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

3. Model Building and Training:

Model Choice: A Support Vector Machine (SVM) model was chosen due to its effectiveness in high-dimensional spaces, like those in the MNIST dataset.
Library: Scikit-learn, a machine learning library in Python, was used for building the SVM model.
Hyperparameters: Appropriate hyperparameters were selected, including the kernel function and regularization parameter, although specific values are not provided in the initial snippets.
Training Process: The SVM model was trained on the training dataset, leveraging scikit-learn's efficient algorithms for fitting the model to the data.2. Data Preprocessing:

```
# Initialize and train SVM classifier
svm_clf = SVC(kernel='rbf', C=10, gamma='scale')
svm_clf.fit(X_train, y_train)

# Save the trained SVM model to disk
with open('model.pkl', 'wb') as f:
    pickle.dump(svm_clf, f)
```

4. Model Training:

Training Process: The SVM model was trained on the training dataset, leveraging scikit-learn's efficient algorithms for fitting the model to the data.

```
# Load the model from disk (optional, to demonstrate model loading)
loaded_model = pickle.load(open('model.pkl', 'rb'))

# Predictions on the test set using the loaded model
y_pred = loaded_model.predict(X_test)
```

5. Model Evaluation:

Evaluation Metrics: The model's performance was evaluated on the testing dataset using metrics such as accuracy, precision, recall, and F1-score.
Results: While specific metrics were not provided in the snippets, the inclusion of these metrics in model.py indicates a comprehensive evaluation approach.

```python
# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Print evaluation metrics
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

6. Model Deployment:

Deployment Environment: The model was deployed in a web application environment, allowing users to interact with the model through a user interface.
Serialization: The trained SVM model was serialized using pickle for use in the web application.

app.py > preprocess_image

You, 33 minutes ago | 1 author (You)

```python
# Piyush - 2021BEC0023
from flask import Flask, render_template, request, jsonify
from PIL import Image
import io
import base64
import numpy as np
import pickle

app = Flask(__name__)

with open("model.pkl", "rb") as f:
    svm_model = pickle.load(f)

# Function to convert base64 string to image
def base64_to_image(base64_string):
    base64_bytes = base64_string.encode('utf-8')
    image_data = base64.b64decode(base64_bytes)
    image = Image.open(io.BytesIO(image_data))
    return image

# Function to preprocess the image for prediction
def preprocess_image(image):    You, 2 hours ago • added flask and pkl file
    # Convert image to grayscale
    image = image.convert('L')
    # Resize image to 28x28 (MNIST dataset size)
    image = image.resize((28, 28))
    # Convert image to numpy array
    image_array = np.array(image)
    # Flatten image array
    image_flattened = image_array.flatten()
    # Normalize image
    image_normalized = image_flattened / 255.0
    return image_normalized

@app.route('/')
def index():
    return render_template('index.html')
```

```python
@app.route('/')
def index():
    return render_template('index.html')


@app.route('/predict_digit', methods=['POST'])
def predict_digit():
    # Get base64-encoded image data from the request
    image_data = request.form['image_data']
    # Convert base64 string to image
    img = base64_to_image(image_data)
    # Preprocess image
    img_processed = preprocess_image(img)
    # Reshape image for prediction
    img_reshaped = img_processed.reshape(1, -1)
    # Predict digit using the trained SVM model
    prediction = svm_model.predict(img_reshaped)[0]
    return jsonify({'prediction': str(prediction)})

if __name__ == '__main__':
    app.run(debug=True)
```

7. User Interfaces:

Interface Features: A simple user interface was developed, allowing users to draw a digit on the screen. This digit image is then sent to the backend for recognition by the SVM model.

templates > index.html > html > head > title

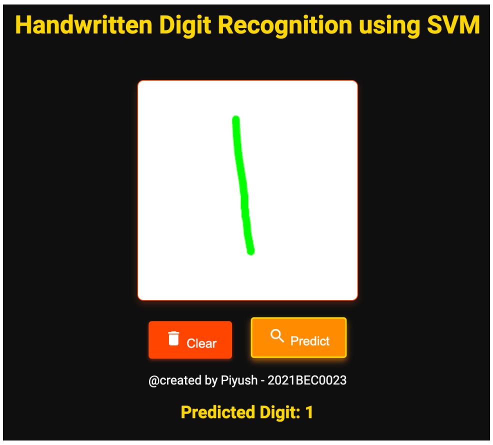
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Digit Recognition by Piyush</title>      You, 34 minutes ago • Uncommitted changes
    <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap" rel="stylesheet">
    <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
    <style>⋯
    </style>
</head>
<body>
    <header>
        <h1>Handwritten Digit Recognition using SVM</h1>
    </header>
    <canvas id="canvas" width="280" height="280"></canvas><br>
    <button onclick="clearCanvas()"><i class="material-icons">delete</i> Clear</button>
    <button onclick="predictDigit()"><i class="material-icons">search</i> Predict</button><br>
    <p>@created by Piyush - 2021BEC0023</p>
    <div id="prediction"></div>
    <script>
        var canvas = document.getElementById('canvas');
        var ctx = canvas.getContext('2d');
        var mousePressed = false;
        var lastX, lastY;

        canvas.addEventListener("mousedown", function (e) {
            mousePressed = true;
            draw(e.offsetX, e.offsetY, false);
        });

        canvas.addEventListener("mousemove", function (e) {
            if (mousePressed) {
                draw(e.offsetX, e.offsetY, true);
            }
        });

        canvas.addEventListener("mouseup", function (e) {
            mousePressed = false;
        });
```

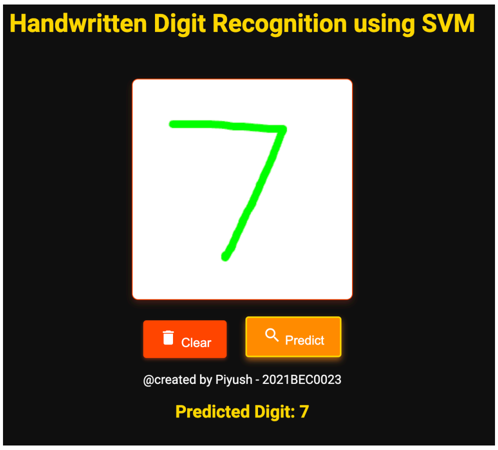
Implementation: The interface was implemented using HTML, CSS, and JavaScript, with Flask serving as the backend to handle requests and serve the model's predictions.

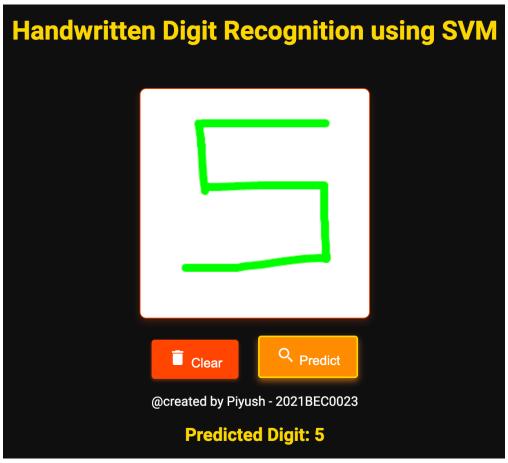
```
    <script>
        var lastX, lastY;

        canvas.addEventListener("mousedown", function (e) {
            mousePressed = true;
            draw(e.offsetX, e.offsetY, false);
        });

        canvas.addEventListener("mousemove", function (e) {
            if (mousePressed) {
                draw(e.offsetX, e.offsetY, true);
            }
        });

        canvas.addEventListener("mouseup", function (e) {
            mousePressed = false;
        });

        canvas.addEventListener("mouseleave", function (e) {
            mousePressed = false;
        });

        function draw(x, y, isDown) {
            if (isDown) {
                ctx.beginPath();
                ctx.strokeStyle = "#0f0"; // Neon stroke color
                ctx.lineWidth = 10;
                ctx.lineJoin = "round";
                ctx.moveTo(lastX, lastY);
                ctx.lineTo(x, y);
                ctx.closePath();        You, 2 hours ago • integrated with fe
                ctx.stroke();
            }
            lastX = x;
            lastY = y;
        }

        function clearCanvas() {
            ctx.clearRect(0, 0, canvas.width, canvas.height);
            $('#prediction').removeClass('show').text('');
        }

        function predictDigit() {
            var canvas = document.getElementById('canvas');
            var imageData = canvas.toDataURL('image/png').replace(/^data:image\/(png|jpg);base64,/, '');
            $.ajax({
                type: "POST",
                url: "/predict_digit",
                data: {image_data: imageData},
                success: function(response) {
                    $('#prediction').text('Predicted Digit: ' + response.prediction).addClass('show');
                },
                error: function(xhr, status, error) {
                    console.error(xhr.responseText);
                }
            });
        }
    </script>
</body>
</html>
```

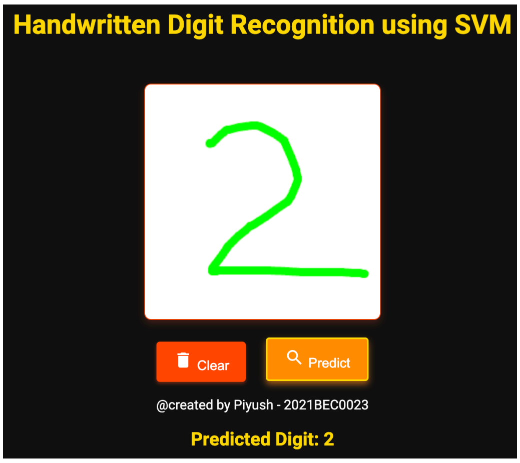
8. Testing :

The final phase of the project entailed a thorough testing process, focusing primarily on the user interface (UI) designed to interact with the SVM-based digit recognition model. This approach allowed for an end-to-end evaluation of the system's performance, encompassing both its technical accuracy and its usability from an end-user perspective.









9. Conclusion :

The digit recognition system demonstrates a successful application of machine learning techniques, from model training with the SVM algorithm to deploying a functional web application. This project not only showcases the practical application of SVM for image recognition tasks but also illustrates the full lifecycle of a machine learning project, including preprocessing, model evaluation, and user interaction through a web interface.