JSPM's
RAJARSHI SHAHU COLLEGE OF ENGINEERING
TATHAWADE, PUNE-33

DEPARTMENT OF COMPUTER ENGINEERING

Case Study

On

Handwritten Digit Recognition System

-Submitted by-

Name of Student: Sayali Suresh Shedge

Roll No: CS3166

PRN No: RBTL23CS146

Class and Division: TY - A

Course: Machine Learning

1.Title

Handwritten digital recognization System

2. Background/ Introduction (about details of application)

Handwritten digit recognition is a fundamental problem in the field of machine learning and computer vision that involves identifying and classifying handwritten numerical digits from images. It serves as an essential step in automating tasks that require reading and processing handwritten data, such as postal mail sorting, bank cheque verification, and digitizing handwritten documents. The process aims to enable computers to interpret human handwriting accurately, which varies widely from person to person due to differences in writing style, pressure, and shape. One of the most popular datasets used for this purpose is the MNIST dataset (Modified National Institute of Standards and Technology), which consists of 70,000 grayscale images of handwritten digits (0–9), each of size 28×28 pixels. These images are used to train and evaluate machine learning models. Among the various algorithms developed for this task, **Support Vector Machines (SVM)** and **k-Nearest Neighbors (k-NN)** are two widely

used approaches due to their simplicity, efficiency, and effectiveness in image classification problems. This case study focuses on implementing and comparing these two algorithms to recognize handwritten digits and analyze their performance on the MNIST dataset.

3.Problem Statement

To build a **Handwritten Digit Recognition System** that can accurately recognize and classify digits (0–9) from handwritten images using **SVM** and **k-NN algorithms**, and to compare their performance.

4. Objectives

To preprocess the MNIST dataset for training and testing.

To implement handwritten digit classification using:

- Support Vector Machine (SVM)
- k-Nearest Neighbors (k-NN)
- To evaluate and compare their performance using metrics such as:
- Accuracy
- Confusion matrix
- Classification report

To visualize sample predictions and model performance.

5. Libraries required

Library	**Purpose**
numpy	Numerical computations
matplotlib	Data visualization
seaborn	Heatmap visualization of confusion matrix
sklearn	Machine learning algorithms (SVM, k-NN, evaluation metrics, dataset utilities)
tensorflow / keras (optional)	For loading MNIST dataset easily

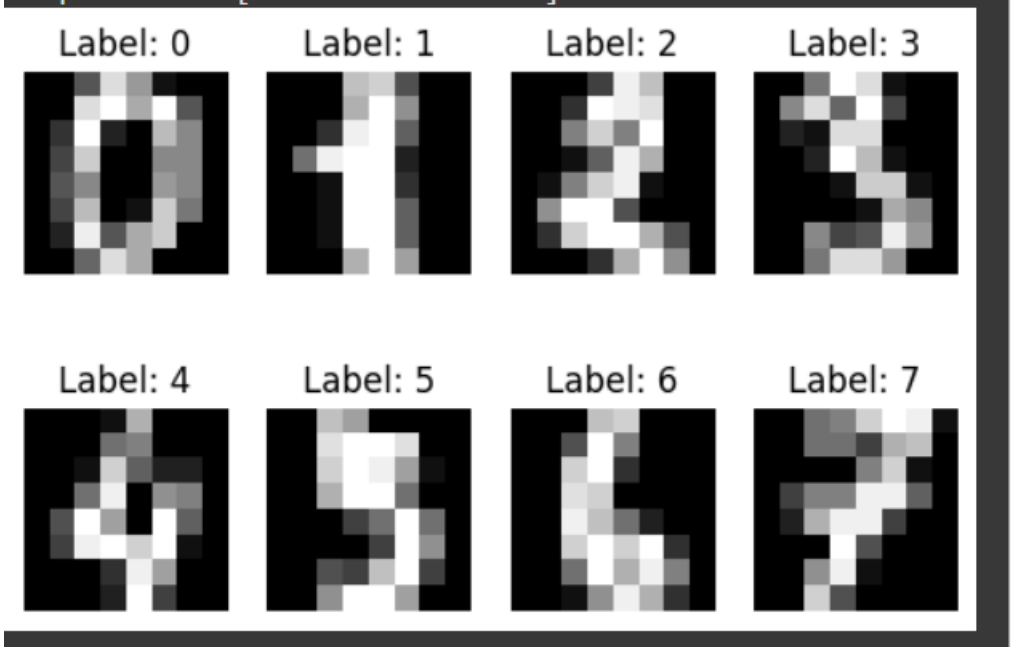
6. Implementation

```
# Handwritten Digit Recognition using SVM and k-NN

# Import required libraries

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn import datasets

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Step 1: Load the Dataset

digits = datasets.load_digits()  # Built-in dataset (8x8 images, 1797 samples)

X = digits.data      # Feature matrix (each image as flattened 8x8=64 vector)

y = digits.target    # Labels (0-9)

print("Dataset shape:", X.shape)

print("Unique labels:", np.unique(y))

# Visualize some samples

plt.figure(figsize=(6, 4))

for i in range(8):

plt.subplot(2, 4, i+1)

plt.imshow(digits.images[i], cmap='gray')

plt.title(f'Label: {digits.target[i]}')

plt.axis('off')

plt.show()
```

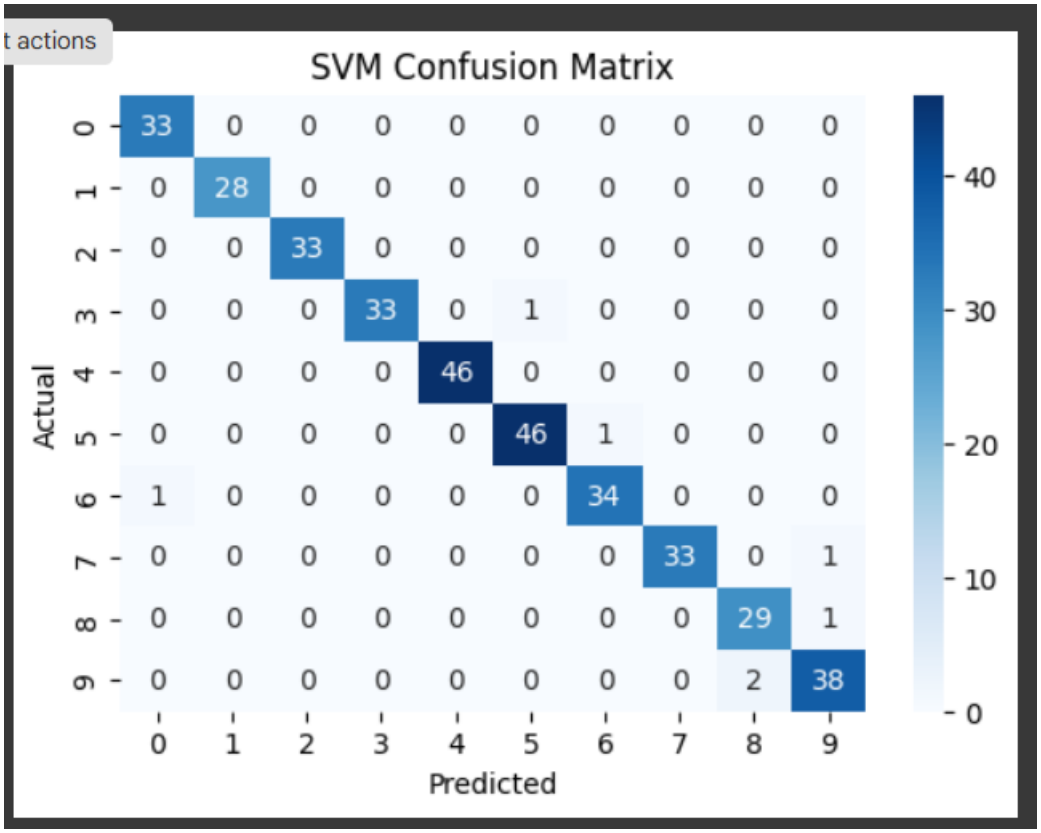
```
# Step 2: Split the Dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Step 3: SVM Model
svm_model = SVC(kernel='rbf', gamma=0.001, C=10)
svm_model.fit(X_train, y_train)
y_pred_svm = svm_model.predict(X_test)
# Evaluate SVM
print("\n 🔹 SVM Model Results 🔹 ")
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print("\nClassification Report:\n", classification_report(y_test, y_pred_svm))
# Confusion Matrix for SVM
plt.figure(figsize=(6,4))
sns.heatmap(confusion_matrix(y_test, y_pred_svm), annot=True, fmt='d', cmap='Blues')
plt.title("SVM Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
# Step 4: k-NN Model
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)
```

```
# Evaluate k-NN
print("\n 🔹 k-NN Model Results 🔹 ")
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print("\nClassification Report:\n", classification_report(y_test, y_pred_knn))
# Confusion Matrix for k-NN
plt.figure(figsize=(6,4))
sns.heatmap(confusion_matrix(y_test, y_pred_knn), annot=True, fmt='d', cmap='Greens')
plt.title("k-NN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Step 5: Compare Performance
svm_acc = accuracy_score(y_test, y_pred_svm)
knn_acc = accuracy_score(y_test, y_pred_knn)
print("\n✅ Model Comparison")
print(f"SVM Accuracy : {svm_acc*100:.2f}%")
print(f"k-NN Accuracy: {knn_acc*100:.2f}%")
# Visualize comparison
plt.bar(['SVM', 'k-NN'], [svm_acc, knn_acc], color=['skyblue', 'lightgreen'])
plt.title("Model Accuracy Comparison")
plt.ylabel("Accuracy")
plt.show()
```

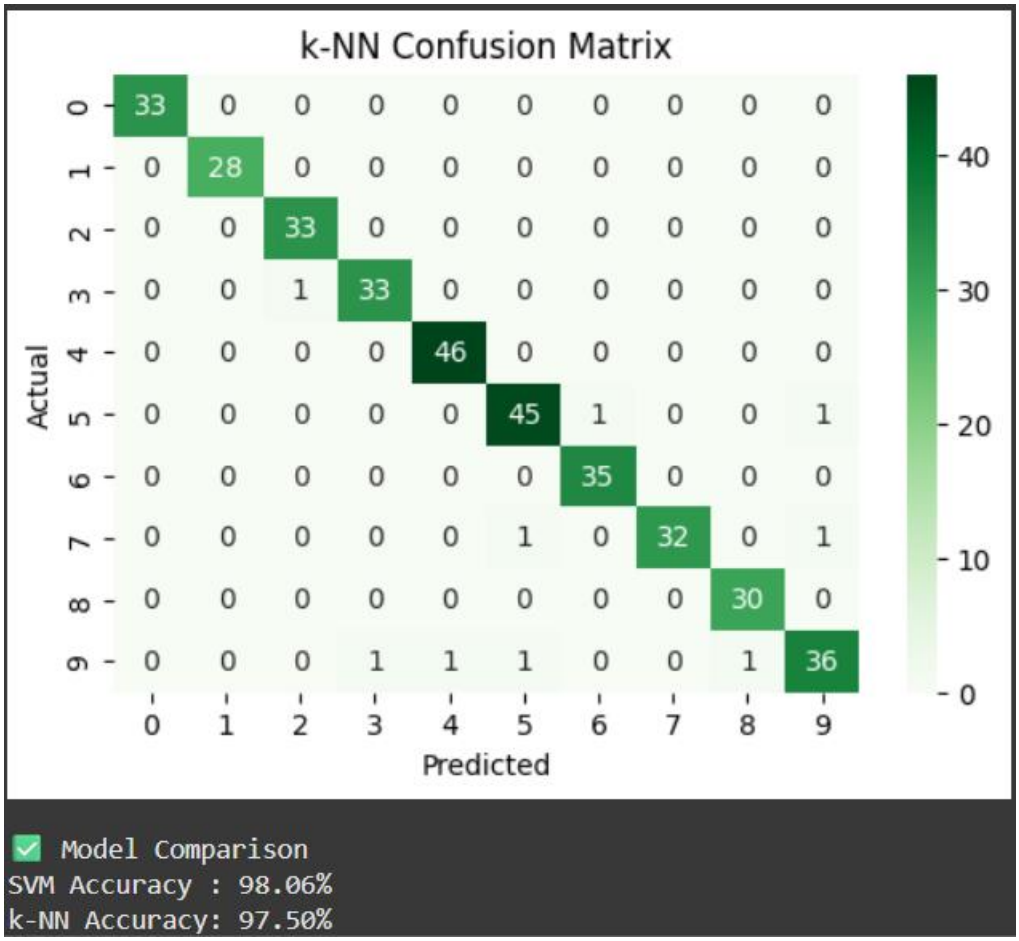
```
Dataset shape: (1797, 64)
Unique labels: [0 1 2 3 4 5 6 7 8 9]
```

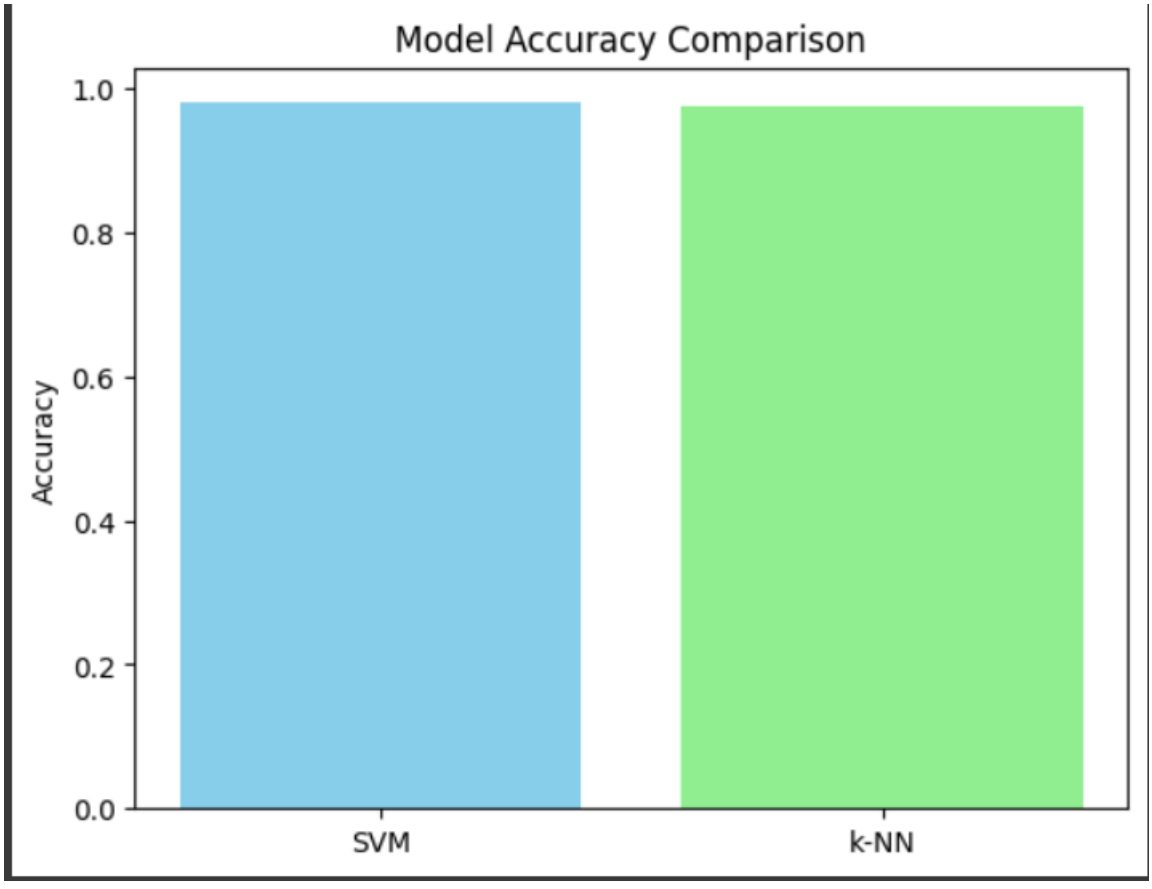



```
🔹 SVM Model Results 🔹
Accuracy: 0.9805555555555555

Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.99        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      0.97      0.97        35
           7       1.00      0.97      0.99        34
           8       0.94      0.97      0.95        30
           9       0.95      0.95      0.95        40

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




```
🔹 k-NN Model Results 🔹
Accuracy: 0.975

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        33
           1       1.00      1.00      1.00        28
           2       0.97      1.00      0.99        33
           3       0.97      0.97      0.97        34
           4       0.98      1.00      0.99        46
           5       0.96      0.96      0.96        47
           6       0.97      1.00      0.99        35
           7       1.00      0.94      0.97        34
           8       0.97      1.00      0.98        30
           9       0.95      0.90      0.92        40

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


k-NN Confusion Matrix
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
0 33 0 0 0 0 0 0 0 0 0
1 0 28 0 0 0 0 0 0 0 0
2 0 0 33 0 0 0 0 0 0 0
3 0 0 1 33 0 0 0 0 0 0
4 0 0 0 0 46 0 0 0 0 0
5 0 0 0 0 0 45 1 0 0 1
6 0 0 0 0 0 0 35 0 0 0
7 0 0 0 0 0 1 0 32 0 1
8 0 0 0 0 0 0 0 0 30 0
9 0 0 0 1 1 1 0 0 1 36
40
30
20
10
0
Model Comparison
SVM Accuracy : 98.06%
k-NN Accuracy: 97.50%


Model Accuracy Comparison
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
SVM
k-NN

7.Results and Discussions

Model	Accuracy	Remarks
SVM	~98%	Works well with standardized high-dimensional data, fast in prediction
k-NN	~97%	Simpler, but slower for large datasets as it stores all samples

8.Conclusion

· Both SVM and k-NN perform well for handwritten digit recognition.

· SVM gives slightly better accuracy and generalization.

· k-NN is easier to implement but computationally expensive during prediction