1. **Problem Statement:MNIST HAND WRITTEN DIGITS**

2. **Practical Applications of this Work:**
    - Postal Sorting Systems: Automating mail sorting by reading handwritten digits.
    - Banking and Check Processing: Automating cheque recognition and verification.
    - OCR Systems: Converting handwritten documents into digital text.
    - Captcha Solving: Recognizing digits in CAPTCHA challenges.
    - Assistive Technologies: Helping visually impaired individuals by converting handwritten digits into speech.
    - Educational Tools: Tracking and improving handwriting in education.
    - Robotics: Enabling robots to read handwritten labels and interact with the environment.
    - Medical Data Digitization: Automating the extraction of handwritten numbers in medical records.

3. **Project Links:**
    a. **Dataset Link: https://www.kaggle.com/competitions/digit-recognizer/data**
    b. **Github Repo:** https://github.com/Bhanusatish2704/HandwrittenDigitRecognizer

4. **Exploratory Data Analysis:**

1. Data Overview

- The MNIST dataset consists of 60,000 training images and 10,000 test images of handwritten digits (0-9), with each image having a size of 28x28 pixels.
- Training set shape: (60000, 28, 28)
- Test set shape: (10000, 28, 28)
- The dataset is divided into training (80%) and validation (20%) sets after flattening the images for training.

2. Visualization

- We visualize a few samples of the handwritten digits to understand the dataset better.

Example of a few images displayed with their labels:
python
Copy code

```
plt.figure(figsize=(10, 5))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(X_train[i], cmap='gray')
    plt.title(f"Label: {y_train[i]}")
    plt.axis('off')
```

```
plt.show()
```

-

3. Data Preprocessing

- Flattening: Images are flattened into vectors of size 784 (28x28) for each image.
- Normalization: Scaling pixel values from 0-255 to 0-1 to improve model performance.
- Shape: Flattened training set shape is (60000, 784), and test set shape is (10000, 784).
- Train/Test Split: A further split of 80/20 is applied for training and validation sets.

4. Model Evaluation

- KNN (K-Nearest Neighbors):
  - KNN model is trained with k=3 neighbors.
  - Validation Accuracy: ~97.27%
  - Confusion Matrix: Confirms correct digit predictions with high precision.
  - Classification Report: Shows metrics such as precision, recall, and F1-score for each class.

  python
  Copy code
  ```
  knn_model = KNeighborsClassifier(n_neighbors=3)
  knn_model.fit(X_train_split, y_train_split)
  y_val_pred_knn = knn_model.predict(X_val)
  ```

-
- SVM (Support Vector Machine):
  - SVM model uses a linear kernel.
  - Validation Accuracy: ~93.52%
  - Confusion Matrix: Highlights correct and incorrect predictions.
  - Classification Report: Shows metrics for each class, with a macro and weighted average.

  python
  Copy code
  ```
  svm_model = SVC(kernel='linear')
  svm_model.fit(X_train_split, y_train_split)
  y_val_pred_svm = svm_model.predict(X_val)
  ```

-
- Test Accuracy:
  - KNN Test Accuracy: ~97.2%
  - SVM Test Accuracy: ~93.5%

5. Confusion Matrices

- Both models (KNN and SVM) have confusion matrices to display misclassified digits.
- KNN Confusion Matrix: Using `sns.heatmap()` for visualization.
- SVM Confusion Matrix: Similarly, SVM confusion matrix can be visualized using `sns.heatmap()`.

Example:

python
Copy code

```
sns.heatmap(confusion_matrix(y_val, y_val_pred_knn), annot=True,
fmt='d', cmap='Blues')
plt.title("KNN Confusion Matrix")
plt.show()
```

6. Key Insights

- KNN Model: Higher accuracy (~97%) compared to SVM. Better suited for digit classification with fewer support vectors.
- SVM Model: Slightly lower accuracy (~93.5%), but still strong with the ability to handle high-dimensional data effectively.
- Classification Performance: Both models perform well in terms of precision, recall, and F1-score for most digits, with minor misclassifications in some numbers like '2', '5', and '8'.

7. Further Steps

- Exploring more advanced models like Deep Learning (CNNs) for improved accuracy.
- Hyperparameter tuning for both KNN and SVM models to further enhance performance..

5. **Data Preprocessing techniques:**
   Reshape: Flatten 28x28 images to 784-length vectors.
   Normalize: Scale pixel values to the [0, 1] range.
   Train/Test Split: Split data into training and test sets (80/20).
   One-Hot Encoding: Convert class labels to one-hot encoded format.
   Augmentation (for CNN): Rotate, zoom, and shift images to create diverse training data.
   Class Balancing: Adjust for class imbalance by oversampling or using class weights.
   Noise Reduction: Apply filters to smooth images and reduce noise.
   PCA: Reduce dimensionality to speed up training (optional).
   Shuffling: Shuffle data to avoid order-related biases.

6. **ML Models Applied:**

1. K-Nearest Neighbors (KNN)

- Accuracy (Validation): ~97.27%
- Description: Classifies based on the majority label of k nearest neighbors. Simple but computationally expensive during prediction.

2. Support Vector Machine (SVM)

- Accuracy (Validation): ~93.52%
- Accuracy (Test): ~93.51%
- Description: Finds the optimal hyperplane to separate classes. Effective in high-dimensional spaces, but can be computationally intensive.

3. Logistic Regression

- Accuracy (Validation): ~91.8% (approx.)
- Description: A linear classifier that works well for simple classification tasks, though it may not perform as well on more complex datasets like MNIST compared to KNN or SVM.

7. **BEST MODEL APPLIED :**

In summary, KNN achieved the highest validation accuracy (~97%), followed by SVM (~93%), and Logistic Regression (lower, ~91%).

8. **Table Comparing Performance of different models:**

Model	Validation Accuracy	Test Accuracy	Precision	Recall	F1-Score	Notes
K-Nearest Neighbors (KNN)	~97.27%	N/A	~97%	~97%	~97%	Simple, easy to implement; computationally expensive for large datasets.
Support Vector Machine (SVM)	~93.52%	~93.51%	~93-95%	~90-97%	~91-94%	Effective in high-dimensional space, but computationally expensive.
Logistic Regression	~91.8%	N/A	~90-92%	~90-92%	~90-92%	Less effective for MNIST compared to KNN and SVM. Linear model.

9. **Best model and why :**

K-Nearest Neighbors (KNN)

- **Validation Accuracy**: ~97.27%
- **Why it's the best**:
    - **Highest Accuracy**: KNN achieved the highest validation accuracy compared to other models (SVM and Logistic Regression).
    - **Simplicity**: Easy to implement and understand; makes predictions based on the majority class of k nearest neighbors.
    - **Interpretability**: The decision-making process is transparent—based on the proximity of data points.
    - **Suitability for MNIST**: Works well with MNIST, as it doesn't require complex training, and performs well on the dataset's relatively simple structure.
- **Limitations**:
    - **Computational Cost**: KNN can be slow during prediction time as it calculates distances to all training data points.
    - **Sensitive to Feature Scaling**: Requires proper normalization (e.g., scaling pixel values) to perform optimally.

10. **Extension and Future Work:**

Future work includes optimizing model hyperparameters, exploring deep learning models like CNNs for improved accuracy, and testing on diverse datasets. Additionally, integrating hybrid approaches, real-time applications, and feature engineering techniques can enhance performance and scalability.