Assignment 3: Training a Scikit-learn Model

Dataset Overview

For this project, I used the Handwritten Digits 10k dataset from HuggingFace. It contains 10,000 images of handwritten digits (0-9) with corresponding labels. However, several preprocessing steps were necessary due to dataset inconsistencies:

- **Image Size and Format:** Images had varying dimensions and color formats.
- **Invalid Labels:** Some labels, such as "g", were invalid for digit classification.
- **Cleaning Process:** Images were resized to 40x40 pixels and converted to grayscale for uniformity. Invalid labels were removed, and pixel values were normalized by dividing each by 255 to avoid overemphasis on high-intensity pixels.

Model Selection and Justification

The chosen model was a Support Vector Machine (SVM) with an RBF kernel (SVC from Scikit-learn). The rationale behind this choice includes:

1. **Effective for High-Dimensional Data:** The transformed images (flattened into 1D arrays) create a high-dimensional feature space. SVMs are particularly effective in such scenarios because they find the optimal hyperplane in high-dimensional spaces.
2. **Robustness to Nonlinear Boundaries:** Using the RBF kernel, the SVM can handle nonlinear decision boundaries effectively. This is crucial for digit classification tasks where patterns (like curves in digits) are not linearly separable.
3. **Performance and Efficiency:** SVMs are memory-efficient and generally fast to train on moderate-sized datasets. In fact, SVMs are computationally efficient compared to deep learning models, which makes them suitable for scenarios with limited computational resources like my own.
4. **Memory Efficiency:** Since SVMs rely on support vectors (a subset of training points), they are memory-efficient compared to models like k-Nearest Neighbors that require storing the entire dataset.
5. **Empirical Evidence:** I saw online that SVMs have been widely recognized as reliable for digit recognition tasks, particularly on datasets like MNIST.

The data was split into an 80/20 ratio for training and testing. A validation set was not explicitly used, as this was a straightforward implementation task.

Model Evaluation

Initial results, as shown in trashModel.png, revealed significant overfitting. Upon investigation, it was discovered that the pixel values were not normalized, causing high-intensity pixels to disproportionately influence the model. After normalizing the pixel values, performance improved dramatically. Initially, I had set the gamma=0.01 parameter arbitrarily, which resulted in an accuracy of approximately 91.05%. However, after removing the gamma=0.01 parameter

from the SVM model, the performance improved further, as evidenced by the updated results (goodModel.png).

Final Performance Metrics:

- **Accuracy:** 92.40%
- **Precision, Recall, and F1-Score:** Averaged around 92% across all classes, as shown in the classification report and confusion matrix.
- **Confusion Matrix Insights:** The matrix (see confusionMatrix.png) indicates that misclassifications were minimal, with strong diagonal dominance reflecting correct predictions for most classes.

Key Observations

1. **Normalization Impact:** Normalizing pixel values was critical to reducing overfitting and improving generalization.
2. **Model Efficiency:** The SVM model trained efficiently on this dataset, balancing performance and resource usage.
3. **Dataset Cleaning:** Handling inconsistencies (e.g., resizing, removing invalid labels) ensured that the model could focus on meaningful patterns in the data.
4. **Parameter Simplification:** Removing the randomly set gamma parameter allowed the SVM to determine an optimal kernel scale automatically, leading to better generalization. This experience highlighted the importance of understanding hyperparameters, such as gamma, before setting them, as arbitrary values can negatively impact model performance.

Conclusion

This project highlighted the importance of data preprocessing and normalization in machine learning. The SVM model demonstrated strong performance for handwritten digit classification, making it a robust choice for similar tasks. Future improvements could include experimenting with deep learning models (e.g., CNNs) to explore potential gains in accuracy and robustness.