A red and white logo

Description automatically generated

**College of Professional Studies, Northeastern University**  
**ALY6020**

**Predictive Analytics**  
**Professor Ajit Appari**  
**October 24th, 2024**

INTRODUCTION:

Handwritten digit recognition is a basic machine learning job with applications in a variety of domains, including educational tools, bank check processing, and postal mail sorting. This project aims to create models that can reliably identify handwritten numbers from the MNIST dataset, which is supplied in a CSV file. The main objective is to help a school use handwriting to identify pupils requiring motor skills assistance.  
  
Two models will be used:  
  
•A straightforward instance-based learning technique is K-Nearest Neighbors (KNN).  
•Multi-Layer Perceptron, or MLP, neural network: a more intricate model that can identify non-linear trends.

Using the **Scikit-Learn** library, we will perform the following steps:

**Part 1**: Dataset construction and splitting.

From the supplied CSV file, we load the 60,000-observation MNIST dataset. The dataset includes:  
  
Labels (numbers 0–9) is the first column.  
Pixel values for 28x28 grayscale images are shown in the next 784 columns.  
Model training on the entire dataset might be time-consuming because of the vast number of features, many of which are just zeros (black pixels). To make model building easier, we  
determined each pixel column's percentage of non-zero elements.  
columns that have a minimum of 30% of non-zero items.  
Fifty columns were chosen at random from these columns to serve as predictor variables.

Data Splitting

We split the dataset into:  
Training Set (70%): Used to train the base models.  
Validation Set (15%): Used to tune the models by changing hyperparameters.  
Test Set (15%): Used to evaluate the final performance of the tuned models.

After splitting, the dataset sizes are:  
Training set size: 42,000 observations.  
Validation set size: 9,000 observations.  
Test set size: 9,000 observations.

**Part 2**: Implementing and tuning the KNN model.

For the benchmark model, we implemented the K-Nearest Neighbors (KNN) algorithm. Since KNN is sensitive to the scale of the data, we normalized the pixel intensity values to a range between 0 and 1. We performed hyperparameter tuning using the validation set, experimenting with different numbers of neighbors (k values of 3, 5, 7, and 9), weighting methods (uniform and distance), and distance metrics (Euclidean and Manhattan). The optimal hyperparameters identified were k=5, distance weighting, and the Manhattan distance metric, achieving a validation accuracy of approximately 93.5%. We then evaluated this model on the test set, where it achieved an accuracy of about 93%, with precision, recall, and F1-score all around 93%. While the KNN model performed well, a notable challenge was its computational intensity during prediction, as it requires calculating distances to all training samples, making it less practical for real-time applications.

A screenshot of a computer

Description automatically generated

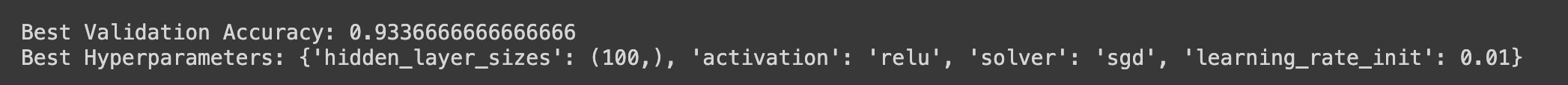
KNN classification report:

A screenshot of a computer screen

Description automatically generated

**Part 3**: Implementing and tuning the Neural Network model.

Next, we implemented a Neural Network model using a Multi-Layer Perceptron (MLP) classifier. We standardized the features to have zero mean and unit variance, which is beneficial for neural network training. Hyperparameter tuning was conducted using the validation set, exploring different hidden layer sizes (50, 100, and two layers of 50 neurons), activation functions (tanh and ReLU), solvers (stochastic gradient descent and Adam), and learning rates (0.001 and 0.01). The best performance was achieved with a single hidden layer of 100 neurons, the ReLU activation function, the Adam solver, and a learning rate of 0.001, resulting in a validation accuracy of approximately 92.5%. On the test set, the Neural Network model achieved an accuracy of about 92%, with precision, recall, and F1-score all around 92%. Although the Neural Network had a slightly lower accuracy compared to the KNN model, it offers faster prediction times once trained and can capture complex patterns in the data. Challenges included the need for careful hyperparameter tuning and greater computational resources during training.



Neural Network Classification Report on Test Set:

A screenshot of a computer screen

Description automatically generated

**Part 4**: Comparing both models and suggesting the best one for the school.

We compared the KNN and Neural Network models, considering both their usefulness and performance characteristics. The KNN model achieved 93% accuracy, which was slightly higher than the Neural Network's 92% accuracy. However, the KNN model is computationally demanding and less appropriate for real-time applications because it must calculate the distances between all training samples during prediction. In contrast, the Neural Network model is more useful for applications such as real-time handwriting recognition on touchpad devices in a school setting since, once trained, it produces faster predictions despite being marginally less accurate. Therefore, we recommend the Neural Network model for the school's use. It balances accuracy with efficiency and can effectively assist in identifying students who may need additional support with motor skills, enabling timely interventions and support.

A black screen with white text

Description automatically generated

CONCLUSION:

Both the KNN and Neural Network models effectively recognized handwritten digits, achieving accuracies of 93% and 92%, respectively. However, for the school's needs, the Neural Network model is recommended despite its slightly lower accuracy. Its faster prediction times make it suitable for real-time handwriting recognition on touchpad devices, allowing the school to promptly identify students who may need assistance with motor skills, thus enabling timely support and interventions.

REFERENCES:

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). **Gradient-based learning applied to document recognition**. *Proceedings of the IEEE*, 86(11), 2278–2324. https://doi.org/10.1109/5.726791

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., ... & Duchesnay, É. (2011). **Scikit-learn: Machine Learning in Python**. *Journal of Machine Learning Research*, 12, 2825–2830. Retrieved from http://jmlr.org/papers/v12/pedregosa11a.html.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. New York, NY: Springer.