Building a Classifier for MNIST with K-Nearest Neighbors (KNN) and Hyperparameter Tuning

Omid Saberi
Dept. of Industrial Automation
University West
Trollhättan, Sweden
https://orcid.org/0009-0005-3596-9182

***Abstract*—This paper reports the development and optimization of a K-Nearest Neighbors (KNN) classifier for the MNIST dataset, comprising 70,000 images of handwritten digits. The objective is to achieve high classification accuracy by optimizing key hyperparameters, such as the number of neighbors (k) and distance weighting, through grid search with cross-validation. A fine-tuned KNN model reaches an accuracy of 97.12%, surpassing baseline performance. Additionally, the paper compares the KNN model's performance with fine-tuned Stochastic Gradient Descent (SGD) and Random Forest classifiers, evaluating them based on F1-score, precision, recall, confusion matrices, and learning curves. Results indicate that the KNN model delivers superior performance in terms of both accuracy and generalization, with minimal overfitting. The Random Forest model also shows strong results, while the SGD classifier demonstrates lower overall performance. This study provides valuable insights into the effectiveness of KNN in multiclass image classification and highlights the importance of hyperparameter tuning in improving model performance.**

***Index Terms*—K-Nearest Neighbors, Hyperparameter Tuning, MNIST, Grid Search, Classifier Performance, Machine Learning, Multiclass Classification.**

I. Introduction

The field of machine learning has revolutionized the way we approach complex pattern recognition tasks, with image classification standing out as a particularly fascinating and challenging domain. Among the various algorithms available, some are prized for their simplicity and effectiveness in handling such tasks. One such method is the K-Nearest Neighbors (KNN) classifier, a simple yet powerful approach that has demonstrated remarkable performance in image classification tasks.

In this assignment, we aim to implement and optimize a KNN classifier for the MNIST dataset, a widely used benchmark in handwritten digit recognition composed of 70,000 images. Our primary objective is to achieve an accuracy exceeding 97% through careful hyperparameter tuning, exploring the effectiveness of this approach in tackling complex image classification problems.

II. KNN Classifier

The K-Nearest Neighbors (KNN) classifier is a lazy learning algorithm that classifies a new instance based on the majority class of its k-nearest neighbors in the feature space. The KNN classifier relies on a distance metric (e.g., Euclidean distance) to determine neighbors' proximity to the query point. The decision-making process is simple.

- Determine the nearest neighbors by calculating the distance between the query point and each point in the training set.
- Assign a class label to the query point by majority voting or weighted voting based on the neighbors' class labels.

Since the algorithm stores the training instances and defers computation until classification, it is called a lazy learner. In KNN, distance weighting can also be applied, where nearer neighbors are given more importance than farther ones using weights inversely proportional to distance. However, the hyperparameter K itself puts a weight on the neighbors beforehand. This means that It gives a weight of zero to every neighbor further than the K nearest ones.

A. Training Process of KNN Classifier

KNN does not have an explicit training phase like some other machine learning models. Instead, it uses the entire training set at the time of prediction, computing the distance from each point in the training set to the query point at classification time. [1].

B. Hyperparameter Tuning using Grid Search

In KNN, key hyperparameters to optimize are as follows.

a) n_neighbors: The number of nearest neighbors (k) used to make the classification.

b) weights: This parameter can either be uniform (where all neighbors contribute equally) or distance-based (where closer neighbors have more influence).

To optimize these parameters, Grid Search with cross-validation is commonly used. In this method, different values for n_neighbors and weights are tested over a grid of possible values. Cross-validation ensures that model performance is evaluated robustly across various train-test splits. Your code implements grid search using GridSearchCV to find the best hyperparameters based on the f1_macro score, which is ideal for multiclass problems [1].

C. *How Hyperparameters Are Optimized*

a) n_neighbors: The goal of tuning n_neighbors is to balance bias and variance. A smaller value of k captures local patterns but may be sensitive to noise, while a larger k reduces variance but risks oversmoothing the classification boundaries.

b) weights: Weighting is optimized by considering the influence of neighbors based on their distance. The distance option ensures that nearer neighbors have a stronger impact on the classification, as described in the article [1].

D. *Code Overview*

The code starts with a basic KNN classifier and uses stratified cross-validation (StratifiedKFold) to evaluate model performance. It uses the confusion matrix, precision, recall, and f1_score as evaluation metrics. It then performs Grid Search to optimize the hyperparameters n_neighbors and weights [2]. This approach effectively covers the fundamental aspects of training and tuning a KNN model.

III. Results and Analysis

To compare the performance of the fine-tuned KNN classifier with the fine-tuned Stochastic Gradient Descent (SGD) and Random Forest models, the F1-score, precision, and recall metrics, alongside confusion matrices and learning curves will be analyzed where applicable.

A. *KNN Classifier Performance*

The fine-tuned KNN classifier achieved the highest performance among the models, with an F1-score of 97.14% on the training set and 97.12% on the test set. Precision and recall also remained similarly high, reflecting that the fine-tuning process was successful in improving the initial KNN performance (96.91% F1-score) with default parameters. As shown in Fig. 1, the confusion matrix for the fine-tuned KNN model shows a high diagonal dominance, suggesting strong accuracy with minimal misclassification. However, slight confusion persists between certain classes, particularly classes like 3 and 9.

The learning curve for the fine-tuned KNN demonstrates convergence with high training and validation scores, suggesting minimal overfitting and optimal model performance given the dataset size (as seen in Fig. 2). Further adjustments to hyperparameters like n_neighbors or weight options could yield marginal improvements.

B. *SGD Classifier Performance*

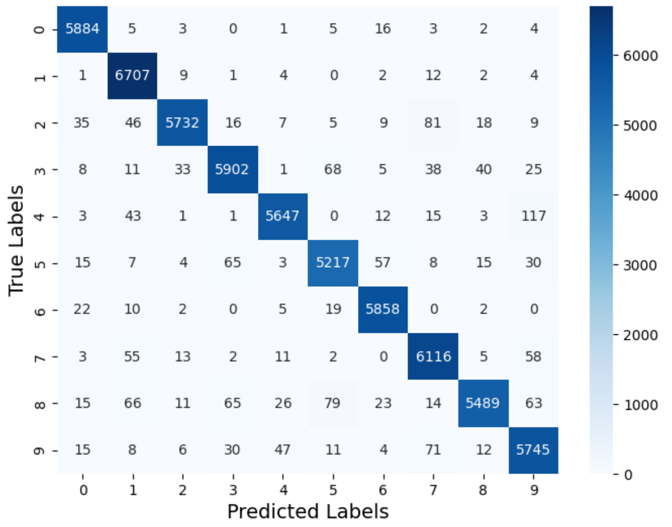
The fine-tuned SGD classifier achieved significantly lower performance, with an F1-score of 86.03%. This indicates that while the model performs decently, it is less effective at handling the multiclass nature of the MNIST dataset compared to the KNN and Random Forest classifiers. The confusion matrix for the fine-tuned SGD (Fig. 3) shows greater misclassification, especially for classes like 0 and 3, where there is notable confusion with neighboring classes.

The learning curve for the fine-tuned SGD, illustrated in Fig. 4, reveals signs of overfitting, as the training score increases rapidly while the validation score remains stable. This suggests that while the model fits well on the training data, it struggles to generalize to new data, leading to a performance plateau.

C. *Random Forest Classifier Performance*

The fine-tuned Random Forest model achieved performance nearly on par with the KNN classifier, with an F1-score of 97.03%. The confusion matrix, depicted in Fig. 5, exhibits similar patterns to KNN, showing strong diagonal dominance and low off-diagonal values, indicating good classification accuracy. Misclassifications are few, although there remains some confusion between closely related classes such as 8 and 9.

The learning curve for the Random Forest model, in Fig. 6, shows strong generalization, similar to KNN, with minimal



	0	1	2	3	4	5	6	7	8	9
0	5884	5	3	0	1	5	16	3	2	4
1	1	6707	9	1	4	0	2	12	2	4
2	35	46	5732	16	7	5	9	81	18	9
3	8	11	33	5902	1	68	5	38	40	25
4	3	43	1	1	5647	0	12	15	3	117
5	15	7	4	65	3	5217	57	8	15	30
6	22	10	2	0	5	19	5858	0	2	0
7	3	55	13	2	11	2	0	6116	5	58
8	15	66	11	65	26	79	23	14	5489	63
9	15	8	6	30	47	11	4	71	12	5745

Fig. 1. KNN Confusion Matrix.

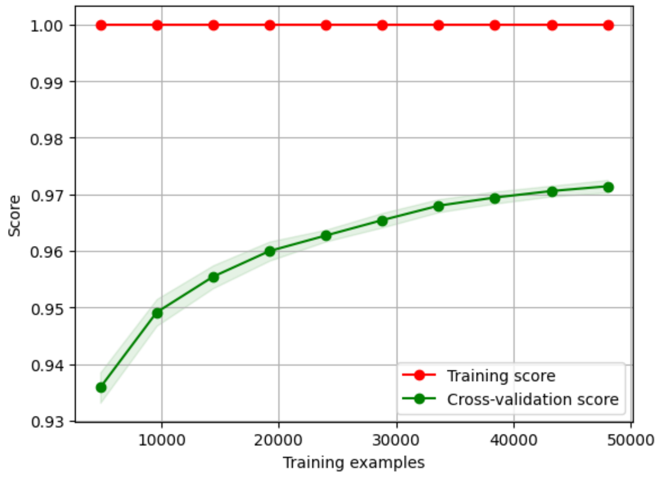


Fig. 2. The KNN Classifier Learning Curve.

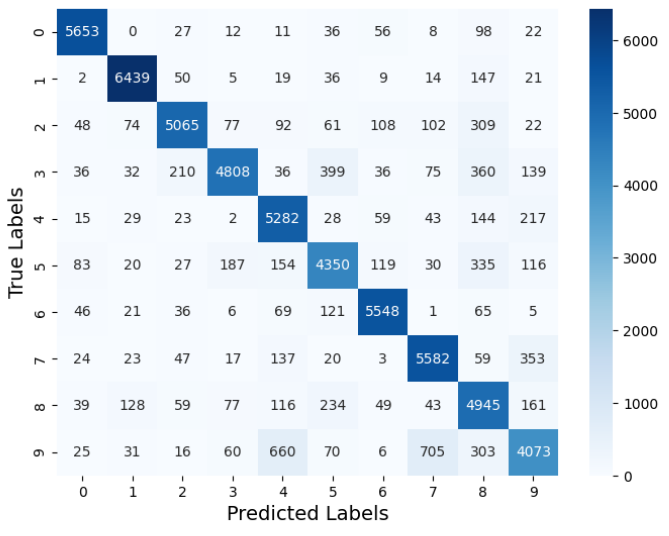


Fig. 3. The SGD Classifier Confusion Matrix.

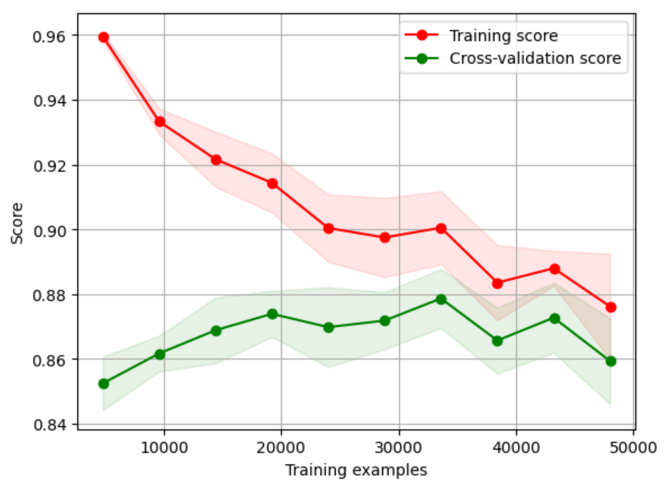


Fig. 4. The SGD Classifier Learning Curve.

overfitting. Both the training and validation scores remain consistently high, demonstrating that this model performs well on both the training and test datasets.

IV. Conclusion

In summary, the fine-tuned KNN and Random Forest classifiers performed similarly well, with both achieving over 97% F1-score on the test set, while the SGD classifier lagged behind with an F1-score of 86.03%. The confusion matrices indicate that KNN and Random Forest models effectively handle class imbalances and perform strong generalization to unseen data. The learning curves for both KNN and Random Forest suggest minimal overfitting, while the SGD model exhibited some signs of overfitting, highlighting areas for potential improvement.

For tasks requiring high accuracy and low misclassification, the KNN and Random Forest classifiers are preferable, with KNN slightly edging out due to higher precision and recall

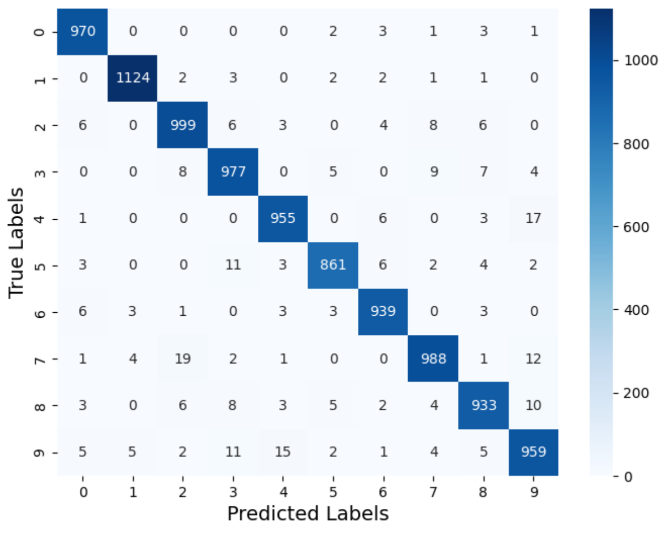


Fig. 5. The Random Forest Classifier Confusion Matrix.

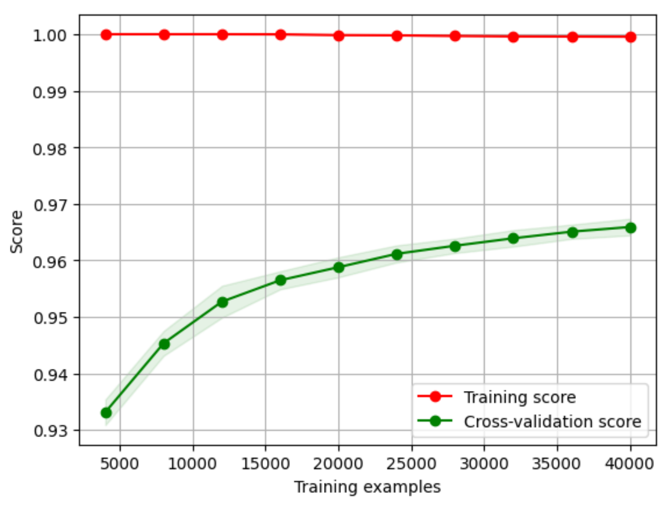


Fig. 6. The Random Forest Classifier Learning Curve.

in some classes. However, Random Forest's robustness to overfitting makes it a competitive alternative.

References

[1] P. Cunningham and S. J. Delany, "k-Nearest Neighbour Classifiers," University College Dublin, 2007.
[2] O. Saberi, "Building a Classifier for MNIST with K-Nearest Neighbors (KNN) and Hyperparameter Tuning," IAI600Lab4, GitHub, 2024. [Online]. Available: https://github.com/omisa69/IAI600Lab4. [Accessed: Oct. 10, 2024].
[3] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," 2013, arXiv:1312.6114. [Online]. Available: https://arxiv.org/abs/1312.6114