

Week 6 Lab

Part A_1:

a)

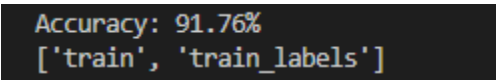
Objective: The objective was to develop a script named `knn_digits_vanilla.py` to train a K-Nearest Neighbors classifier on the MNIST-like dataset of handwritten digits. The goal was to evaluate the classifier's performance with a 50%/50% train/test split and set $k=5$ for the KNN algorithm.

Code Description: First, the digits were loaded from an image file containing 5000 hand-drawn digits (50 rows & 100 columns, each digit being 20x20 pixels).

Then the image was converted to grayscale to simplify the handling of pixel intensity data and the digits were extracted into a 4-dimensional NumPy array and then reshaped into two sets: training and testing datasets, each consisting of 2500 samples.

The labels were generated for each class (0 through 9) and replicated for each sample in the training and testing sets. A KNN model was instantiated and trained using the prepared training data and corresponding labels. The trained model was then tested using the test dataset. The accuracy was calculated by comparing the predicted results against the actual test labels.

Conclusion: The KNN classifier achieved an accuracy of approximately 91.52%, indicating a high level of performance in recognizing handwritten digits from the dataset.



```
Accuracy: 91.76%  
['train', 'train_labels']
```

Figure 1. Results

b)

Objective: In this part of the project, we focused on evaluating the performance of a K-Nearest Neighbors (KNN) classifier across a range of k values from 1 to 9 to understand the impact of k on digit recognition accuracy. The task was implemented in a script named `knn_digits_k_set_tr_50.py`, where the train/test data split was maintained at 50%/50%.

Code Description: The handwritten digits dataset, composed of 5000 images arranged in a grid of 50 rows and 100 columns, was used for this experiment. Each image, representing a single digit, was 20x20 pixels in size. This dataset was converted to grayscale to facilitate the processing and split into individual cells to create arrays suitable for KNN classification.

For each value of k from 1 to 9, a KNN model was trained on the training data and then tested on the test data. The labels for training and testing were numerically encoded to represent each of the ten digit classes.

The classification accuracy was calculated by comparing the predicted results against the actual test labels for each k . The results showed variations in accuracy as k increased, highlighting the sensitivity of KNN performance to the choice of k .

Lastly, the accuracies for each k were plotted in a graph to visually represent the performance trend. This graph served as a critical tool for analyzing the optimal k value that balances bias and variance effectively.

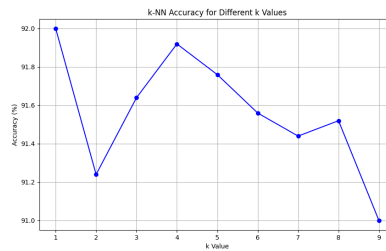


Figure 2. knn_digits_k_set_tr_50 results

Conclusion: Overall, this experiment provided useful information to understand the behavior of KNN classifiers in digit recognition tasks and demonstrated how the choice of k influences the model's ability to generalize from training data to unseen test data. The plot of accuracy against k values demonstrated the practical implications of parameter tuning in ML algorithms.

c)

Objective: In this part of the project, we expanded our exploration of the K-Nearest Neighbors (KNN) classifier by examining its performance across all possible combinations of training/testing splits and k values from 1 to 9.

Code Description: A standard dataset of handwritten digits, formatted into a grid and split into individual cells, each representing a 20x20 pixel image of a digit, was used. These cells were reshaped into a flat array of 400 features per sample and normalized for the purpose of training.

For each train/test split ranging from 10%/90% to 90%/10%, the number of training and testing samples for each class was meticulously calculated to ensure a balanced representation of all digit classes across the different splits. For each split, a KNN model was trained for each k value from 1 to 9.

Classification accuracy for each combination of k and split was recorded, allowing for an observation of the influence of both the size of the training set and the complexity of the model (as varied by k) on the classifier's performance.

The results can be visualized in the graph below which shows the classification accuracy against k values for each training/testing split. This plot can be used for quick identification of trends and optimal combinations of k and training split percentages that yield the highest accuracy.

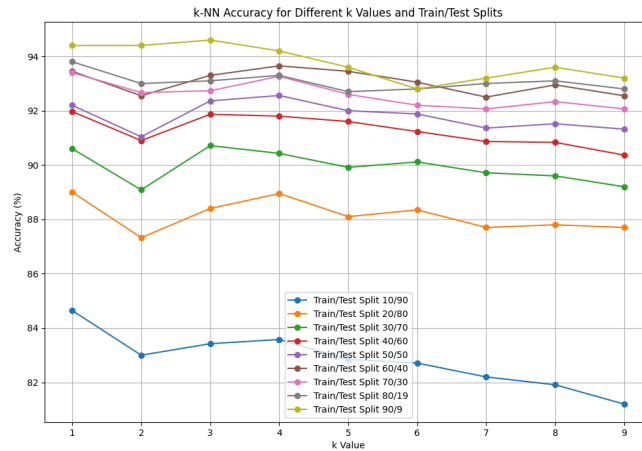


Figure 3. knn_digits_k_set_tr_set results

Conclusion: This extensive testing not only validated the robustness of the KNN classifier under various conditions but also provided a better understanding into the trade-offs between training data availability and model complexity, relevant for practical applications of ML models.

- d) Objective:** In this part of the project, the analysis is extended to the English Alphabet dataset to evaluate the performance of the K-Nearest Neighbors (KNN) classifier with different combinations of training/testing splits and values of k . The process involved in this analysis is similar to the one used for the digits dataset, emphasizing the model's adaptability to different types of data. This exploration is encapsulated in the script `knn_alphabet_k_set_tr_set.py`.

Code Description: The dataset used includes features and labels of English alphabet characters. After fetching the dataset, the features are extracted and prepared for the classifier. Labels are encoded from their original string format into numerical values using the `LabelEncoder` for compatibility with the KNN classifier.

The study spans across training/testing splits ranging from 10%/90% to 90%/10%, reflecting a comprehensive examination of the classifier's robustness and flexibility under varying amounts of training data. For each split configuration, a KNN model is trained and tested with k values from 1 to 9, providing insights into the optimal k value for each split scenario.

Classification accuracy is meticulously recorded for each combination of k and training/testing split. These accuracies are then visualized in a plot, facilitating an easy comparison across the different configurations.

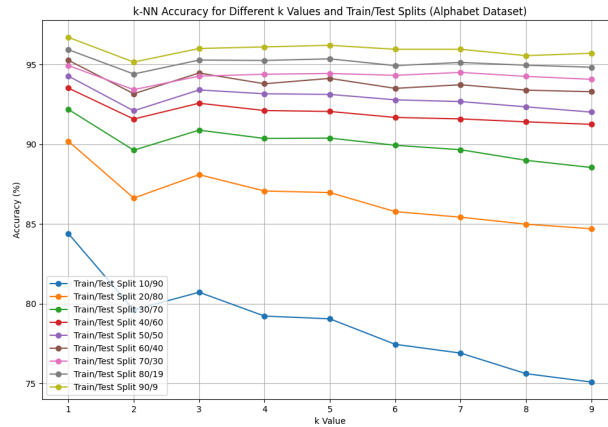


Figure 4. knn_digits_k_set_tr_set results

Conclusion: The plot above demonstrates and highlights the impact of the training set size and the k value on the model's performance, offering valuable insights into the trade-offs involved in model training.

e)

Objective: For this task, color quantization was applied to an image using the K-Means clustering algorithm provided by OpenCV. This process involves reducing the number of colors in an image, which can be useful for various applications such as image compression or simplifying the color palette for analysis.

Code Description: For this task, color quantization was applied to an image using the K-Means clustering algorithm provided by OpenCV. This process involves reducing the number of colors in an image, which can be useful for various applications such as image compression or simplifying the color palette for analysis.

The process started by loading the image nature.png and reshaping it into a two-dimensional array where each row represents a pixel and the columns represent the RGB color components. The pixel values were then converted to np.float32 type, as required by the cv.kmeans() function.

The criteria for K-Means clustering were set to terminate the algorithm either after 10 iterations or when the cluster centers have moved less than epsilon (1.0 in this case), whichever came first. K-Means was executed with different values of K (number of clusters) specified as [2, 3, 5, 10, 20, 40]. This range was chosen to explore the effects of different levels of color quantization on the image, from very coarse to more refined representations.

For each value of K, the K-Means algorithm was applied to find the cluster centers (dominant colors) and to assign each pixel to the nearest cluster. The output image was reconstructed by replacing each pixel's color with the corresponding cluster center's color, effectively reducing the color palette of the image.

Color Quantization with Different K values

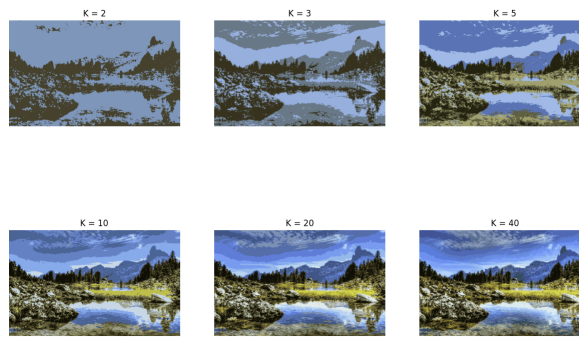


Figure 5. color_quantization results

Conclusion: Each subplot above displays the quantized image with its respective K value, showing how the image appearance changes as more clusters are used. The color differences became less noticeable as K increased, illustrating the trade-off between color diversity and quantization.