Analyzing the Effectiveness of Different Algorithms on Classification of Handwritten Digits

Weston Hawes
California State University, Fresno
May 8, 2023

Abstract

This report aims to investigate the effectiveness of various algorithms in classifying handwritten digits. The algorithms analyzed in this report include k-NN, k-means clustering, MLP, and CNN. Each algorithm is introduced, and its specific use in classification problems is discussed. Supervised and unsupervised learning are explained, and the importance of confusion matrices in evaluating the accuracy of models is highlighted. The MNIST dataset is used for training and testing the models, and a detailed breakdown of the MATLAB code implementation is provided. The results show that the CNN model outperforms the other models, achieving a validation accuracy of 96.00% while the K-Means algorithm performs the worst due to using low k values. Overall, this report provides insights into the world of classification algorithms and their efficacy in solving real-world problems.

1 Introduction

The motivation for this report is to analyze the effectiveness of different algorithms on the classification of handwritten digits. In this report, we will examine several algorithms including k-NN, k-means clustering, multilayer perceptron (MLP), and convolutional neural networks (CNN), and their efficacy in classifying handwritten digits. We will also provide a brief background on each algorithm and highlight its specific use in classification problems. Additionally, we will explain supervised and unsupervised learning and discuss the importance of confusion matrices in assessing the accuracy of our models. Lastly, we will provide a detailed breakdown of our MATLAB code implementation and discuss the dataset used in this report, MNIST. Let's delve into the world of classification algorithms and assess their effectiveness in solving real-world problems.

2 Background Material

Key concepts utilized in this project will be defined and linked to this project in this section.

2.1 K-NN

K-NN (K-Nearest Neighbors) is a machine learning algorithm that utilizes unsupervised learning; assumes little about the dataset. Instead, it searches for the K closest data points in the training set to a given test data point and assigns a label based on the majority class of those neighbors.

2.2 K-Means

K-Means is an unsupervised learning clustering algorithm K-Means aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean. K-Means does this by minimizing the sum of squared distances between the cluster center and each data point.

2.3 MLP

MLP (Multilayer Perceptron) is a form of a neural network, and it is heavily used in classification and regression tasks. MLP is made up of multiple layers of nodes that are connected. The layers can be classified as an input layer, a hidden layer, and an output layer. Each of the nodes utilizes a non-linear activation function for its input, this allows complex relationships to emerge between the inputs and outputs.

2.4 CNN

CNN (Convolutional Neural Network) is a form of a neural network that specializes in processing and image recognition tasks. It works by applying a series of convolutional filters to the input image, which can detect features such as edges and corners, and then using pooling layers to reduce the dimensionality of the output.

2.5 Supervised Learning

Supervised Learning is a form of machine learning where the data points are linked with known outcomes and the algorithm is trained on a labeled dataset. The purpose of this is to learn a route from input to output by minimizing the difference between the true output and the predicted output. Doing so will allow new predictions on new, unanalyzed data to be made from the algorithm.

2.6 Unsupervised Learning

Unsupervised Learning is a form of machine learning where the data points are not yet linked to any sort of categorization, this means that the algorithm is trained on an unlabeled dataset. The purpose of this is to find the pattern and context of data without necessarily categorizing the data.

2.7 Confusion Matrices

Confusion matrices are essentially a table that summarizes true positives, true negatives, false positives, and false negatives generated by the model's predictions. It is essentially a table that summarizes true positives, true negatives, false positives, and false negatives generated by the model's predictions. These values help compute various performance metrics like precision, recall, and accuracy which allows more information on the model's efficacy.

3 Approach

3.1 Dataset Used for Training

The data set used in this report is called "MNIST" and it was created by Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. It was first released in 1999, with a modified version released in 2003. MNIST is a collection of handwritten digit images, commonly used for training and testing image processing and machine learning algorithms. The data set consists of 60,000 training images and 10,000 testing images, each of which is a 28x28 grayscale image of a handwritten digit (0-9). This data set can be downloaded from the official MNIST website: **http://yann.lecun.com/exdb/mnist/.**

3.2 Code Breakdown

This MATLAB code has five main sections, initialization of data, implementing a k-NN classifier, implementing a k-means clustering classifier for supervised classification, training a multilayer perceptron (MLP) classifier, and training a convolutional neural network (CNN) classifier.

For the data initialization:

1. Load the MNIST dataset using the **load** function. This loads the variables **training** and **test**, which contain the training and test data, respectively.
2. Set the number of training, validation, and test examples per class using the variables **num_train**, **num_val**, and **num_test**, respectively.
3. Reshape and convert the training and test images to double using the **reshape** and **double** functions, respectively.
4. Initialize variables to hold the training, validation, and test data and labels.
5. Loop over all classes (0-9) and for each class:
    a. Get the indices of the training, validation, and test examples for this class.
    b. Select **num_train** examples for training, **num_val** examples for validation, and **num_test** examples for testing.
    c. Normalize the data using max-min normalization.
    d. Add the normalized data and labels to the overall training, validation, and test data and labels.

For the k-NN classifier:

1. Implement a k-NN classifier with k = 1, 5, and 10.
    a. For each value of k, loop over all test samples, and for each sample:
        i. Calculate the Euclidean distance between the test sample and all training samples.

        ii. Get the indices of the k nearest neighbors.
        iii. Get the labels of the k nearest neighbors.
        iv. Predict the class of the test sample as the majority class among the k nearest neighbors.
    b. Calculate the confusion matrix and display the accuracy.

For the k-means clustering classifier:

1. Implement a k-means clustering classifier for supervised classification with k = 10, 20, and 30.
    a. For each value of k, train k-means on the training data.
    b. Assign labels to clusters based on the closest training example.
    c. Test on the test data.
    d. Compute the accuracy and the confusion matrix.

For the MLP classifier:

1. A vector named hidden_sizes is defined, which contains the number of neurons in the hidden layers to be trained.
2. For each number of neurons in the hidden layers, the labels are encoded using one-hot encoding.
3. The MLP architecture is defined using the patternnet function.
4. The MLP is trained using the train function with the training data.
5. The MLP is tested using the test data.
6. The accuracy is computed by comparing the predicted labels with the true labels, and then the accuracy is displayed.

For the CNN classifier:

1. The data is split into training, validation, and testing sets.
2. The data is normalized using max-min normalization.
3. The data is converted back to its original 4D shape.
4. The labels are converted to a categorical format.
5. The layers for the CNN are defined, including an image input layer, a convolutional layer, a ReLU activation layer, a max pooling layer, a fully connected layer, a softmax layer, and a classification layer.
6. The options for training the CNN are defined, including the optimization algorithm, the maximum number of epochs, the mini-batch size, and the validation data and frequency.
7. The network is trained using the trainNetwork function with the training data, the defined layers, and the defined options.

4 Results

4.1 K-NN

4.1.1 Confusion Matrix

```
k-NN classifier with k = 1, confusion matrix:
   493     0     1     0     0     2     4     0     0     0
     0   494     0     3     0     0     2     1     0     0
     9    12   447     6     2     0     5    16     2     1
     1     3     2   444     1    23     3    11     7     5
     0     4     0     0   455     0     6     2     0    33
     5     4     0    14     3   447    10     3     6     8
    13     3     1     0     2     2   478     0     1     0
     0    25     1     1     3     1     0   449     0    20
     6     4     9    22     6    15     6     5   416    11
     4     5     1     5    22     4     1    13     5   440
```

Figure 1

```
k-NN classifier with k = 5, confusion matrix:
   491     0     0     0     0     1     7     0     1     0
     0   495     2     2     0     0     1     0     0     0
    11    33   425     6     2     0     5    16     2     0
     0     4     4   450     1    18     1     9     8     5
     0     8     1     0   451     0     7     2     1    30
     5     7     1    11     5   453     5     2     3     8
     8     6     1     0     3     4   478     0     0     0
     0    36     2     1     7     1     0   438     0    15
    13     9     7    19     8    16     6     8   405     9
     3     7     1     8    14     4     1     8     2   452
```

Figure 2

```
k-NN classifier with k = 10, confusion matrix:
   493     0     1     0     0     0     5     0     1     0
     0   495     2     2     0     0     1     0     0     0
     9    50   401     5     2     0     5    22     5     1
     1     9     3   455     1     9     2    11     4     5
     0     9     1     0   451     0     8     2     1    28
     5     8     1     9     8   445     7     1     1    15
    13     7     0     0     5     4   471     0     0     0
     0    41     1     0     7     0     0   436     0    15
    15    11     3    19     8    16     5     9   402    12
     5     9     0     8    14     2     1    10     0   451
```

Figure 3

4.1.2 Accuracy

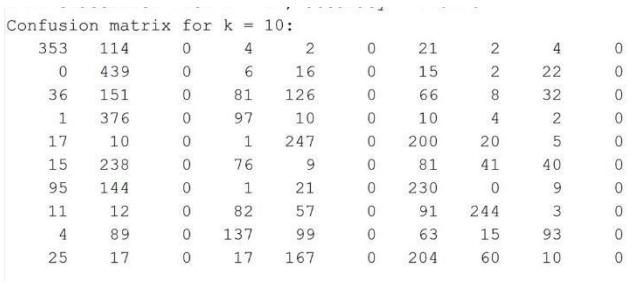
From Figure 1, when k = 1, the accuracy was 91.26%.
From Figure 2, when k = 5, the accuracy was 90.76%.
From Figure 3, when k = 10, the accuracy was 90.00%.

4.1.3 Discussion

The results from the k-means algorithm suggest that as we increase the number of clusters, the accuracy decreases. When k=1, there is only one cluster, and the algorithm can easily identify which cluster each point belongs to, resulting in a high accuracy of 91.26%. However, when k=5 and k=10, there are more clusters, which makes it harder for the algorithm to accurately classify each point, leading to a decrease in accuracy. This is likely because there is more overlap between clusters as the number of clusters increases, making it more difficult to distinguish between them. Therefore, it is important to choose the appropriate number of clusters for a given dataset to achieve the highest accuracy.

4.2 K-Means

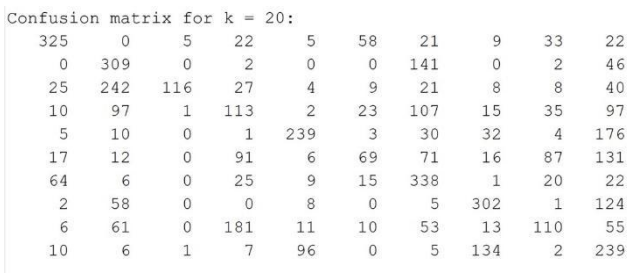
4.2.1 Confusion Matrix



Confusion matrix for k = 10:

353	114	0	4	2	0	21	2	4	0
0	439	0	6	16	0	15	2	22	0
36	151	0	81	126	0	66	8	32	0
1	376	0	97	10	0	10	4	2	0
17	10	0	1	247	0	200	20	5	0
15	238	0	76	9	0	81	41	40	0
95	144	0	1	21	0	230	0	9	0
11	12	0	82	57	0	91	244	3	0
4	89	0	137	99	0	63	15	93	0
25	17	0	17	167	0	204	60	10	0

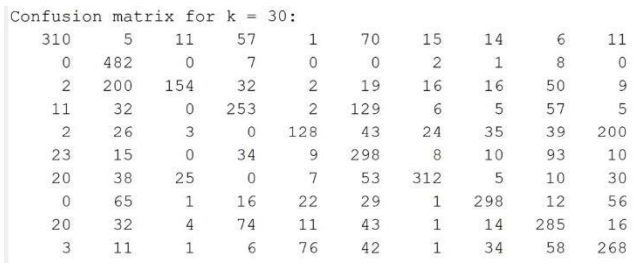
Figure 4



Confusion matrix for k = 20:

325	0	5	22	5	58	21	9	33	22
0	309	0	2	0	0	141	0	2	46
25	242	116	27	4	9	21	8	8	40
10	97	1	113	2	23	107	15	35	97
5	10	0	1	239	3	30	32	4	176
17	12	0	91	6	69	71	16	87	131
64	6	0	25	9	15	338	1	20	22
2	58	0	0	8	0	5	302	1	124
6	61	0	181	11	10	53	13	110	55
10	6	1	7	96	0	5	134	2	239

Figure 5



Confusion matrix for k = 30:

310	5	11	57	1	70	15	14	6	11
0	482	0	7	0	0	2	1	8	0
2	200	154	32	2	19	16	16	50	9
11	32	0	253	2	129	6	5	57	5
2	26	3	0	128	43	24	35	39	200
23	15	0	34	9	298	8	10	93	10
20	38	25	0	7	53	312	5	10	30
0	65	1	16	22	29	1	298	12	56
20	32	4	74	11	43	1	14	285	16
3	11	1	6	76	42	1	34	58	268

Figure 6

4.2.2 Accuracy

From Figure 4, when k = 10, the accuracy was 34.06%.
From Figure 5, when k = 20, the accuracy was 43.2%.
From Figure 6, when k = 30, the accuracy was 55.76%.

4.2.3 Discussion

We can see that the accuracy of the K-Means increased linearly by about 10% per each increase of 10 to k. This could be attributed to K-Means having a poor classification of the data due to the lack of sufficient clusters. The more clusters or k that the algorithm has, the better it can distinguish between different classes.

4.3 MLP

4.3.1 Confusion Matrix

```
Confusion matrix:
[[485   0   1   1   0   2   8   0   3   0]
 [  0 478   3   5   1   0   5   3   5   0]
 [  4   6 433  15   6   2   9  10  14   1]
 [  4   2   8 423   2  33   2  14   8   4]
 [  0   2   2   0 468   0   6   1   5  16]
 [  6   2   1  15   5 429   8  11  18   5]
 [ 10   4   3   1  20   9 442   5   6   0]
 [  1  13  13   6   9   1   1 430   3  23]
 [  7   3   6  22   5  21   3   6 421   6]
 [  6   2   2  15  31   5   1  14  12 412]]
```

Figure 7

```
Confusion matrix:
[[480   0   4   2   0   8   3   1   1   1]
 [  0 486   4   3   1   1   3   1   1   0]
 [  1   7 434  10   7   5   6  15  13   2]
 [  1   0  12 416   1  37   3  15  12   3]
 [  0   2   3   0 443   1  10   2   3  36]
 [  6   1   2  13  10 438   5   6  12   7]
 [ 12   3   9   0  20  20 430   2   3   1]
 [  3   6  16  17   5   2   2 427   1  21]
 [  6   5   6  15   9  17   2   9 421  10]
 [  4  10   1   9  21   4   2  11   7 431]]
```

Figure 8

4.3.2 Accuracy

From Figure 7, when the hidden size = 50, the accuracy was 88.42%.
From Figure 8, when the hidden size = 100, the accuracy was 88.12%.

4.3.3 Discussion

These results suggest that increasing the hidden layer size did not necessarily improve the accuracy of the MLP. The accuracy decreased slightly when the hidden layer size was increased from 50 to 100. It is possible that the additional complexity introduced by the larger hidden layer size led to overfitting, which can result in decreased accuracy on the test set.

4.4 CNN

Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
1	00:00:02	3.91%	8.80%	2.3418	2.3267	0.0100
25	00:00:04	68.75%	76.10%	1.4748	1.2837	0.0100
50	00:00:06	85.16%	87.20%	0.6200	0.4442	0.0100
75	00:00:07	87.50%	89.50%	0.3869	0.3583	0.0100
100	00:00:09	88.28%	91.50%	0.3709	0.3291	0.0100
125	00:00:10	92.19%	90.60%	0.2648	0.3092	0.0100
150	00:00:11	89.84%	91.60%	0.3348	0.2876	0.0100
175	00:00:13	92.19%	92.20%	0.2477	0.2799	0.0100
200	00:00:14	91.41%	92.60%	0.3508	0.2667	0.0100
225	00:00:16	96.09%	92.70%	0.1243	0.2617	0.0100
250	00:00:17	94.53%	92.90%	0.1475	0.2507	0.0100
275	00:00:19	95.31%	93.60%	0.1571	0.2487	0.0100
300	00:00:20	95.31%	93.40%	0.2041	0.2392	0.0100
325	00:00:22	97.66%	93.40%	0.1185	0.2296	0.0100
350	00:00:23	95.31%	93.90%	0.1883	0.2263	0.0100
375	00:00:24	93.75%	93.50%	0.2081	0.2298	0.0100
400	00:00:26	93.75%	93.60%	0.1713	0.2146	0.0100
425	00:00:27	96.88%	93.90%	0.1411	0.2089	0.0100
450	00:00:29	92.97%	94.80%	0.1783	0.2077	0.0100
475	00:00:30	93.75%	94.80%	0.1911	0.1965	0.0100
500	00:00:31	95.31%	94.30%	0.1727	0.2018	0.0100
525	00:00:33	97.66%	94.60%	0.1186	0.1918	0.0100
550	00:00:35	95.31%	95.00%	0.1480	0.1894	0.0100
575	00:00:36	94.53%	94.80%	0.1788	0.1849	0.0100
600	00:00:38	97.66%	94.70%	0.1229	0.1838	0.0100
625	00:00:39	95.31%	95.20%	0.1329	0.1789	0.0100
650	00:00:41	97.66%	94.70%	0.1297	0.1850	0.0100
675	00:00:42	96.88%	95.10%	0.0980	0.1731	0.0100
700	00:00:44	94.53%	95.30%	0.1594	0.1686	0.0100
725	00:00:45	100.00%	95.10%	0.0440	0.1779	0.0100
750	00:00:47	97.66%	95.70%	0.0978	0.1625	0.0100
775	00:00:48	97.66%	95.40%	0.1057	0.1698	0.0100
800	00:00:50	96.88%	95.60%	0.0961	0.1627	0.0100
825	00:00:51	97.66%	95.60%	0.1148	0.1594	0.0100
850	00:00:53	96.09%	96.00%	0.1202	0.1625	0.0100
875	00:00:55	97.66%	95.80%	0.1111	0.1586	0.0100
900	00:00:56	96.88%	95.90%	0.1699	0.1519	0.0100
925	00:00:58	99.22%	95.30%	0.0949	0.1648	0.0100
950	00:00:59	99.22%	95.80%	0.0465	0.1503	0.0100
975	00:01:01	98.44%	96.00%	0.0660	0.1497	0.0100

Figure 9

4.4.1 Confusion Matrix

```
Confusion Matrix:
   490     0     1     0     0     1     6     1     1     0
     0   487     3     2     1     0     3     0     3     1
     2     2   464     9     5     0     4     8     5     1
     2     0     4   474     0     8     1     7     2     2
     0     1     1     0   480     0     4     1     1    12
     4     1     0     5     0   469     5     2    12     2
    11     4     2     0     7     7   461     2     6     0
     0     3    15     4     1     0     0   463     2    12
     6     1     3    13     6     4     1     4   459     3
     3     4     2     2    11     2     0    10     3   463
```

Figure 10

4.4.2 Accuracy

From Figure 10, the CNN was run as in Figure 9, the validation accuracy finished at 96.00%.

4.4.3 Discussion

It is possible that continuing to train the model for more epochs would lead to even higher accuracy on the validation set, although there may also be a risk of overfitting if the model is trained for too long. Overall, the results suggest that the CNN model is a very effective way to classify images and could be a useful tool for a wide range of applications where image recognition is required.

5 Discussion

We identified possible limitations in this project, including the use of a limited dataset and the lack of hyperparameter tuning. To improve this project in the future, expanding the dataset to include more varied handwritten digits and hyperparameter tuning could be incorporated. Additionally, exploring other algorithms, such as decision trees or random forests, could also be investigated for classification tasks. Despite these limitations, this project provides an excellent foundation for understanding and comparing various algorithms' effectiveness in solving real-world classification problems.

6 Conclusion

In conclusion, this report explored the effectiveness of various machine learning algorithms, including k-NN, k-means clustering, multilayer perceptron, and convolutional neural networks, in classifying handwritten digits. We discussed the key concepts and background information related to each algorithm, supervised and unsupervised learning, and the importance of confusion matrices in assessing model accuracy. We then presented a detailed breakdown of the MATLAB code implementation and the MNIST dataset used for training. While all the algorithms performed well, the results showed that convolutional neural networks achieved the highest accuracy in digit classification, followed by multilayer perceptron and k-NN. K-means clustering performed relatively poorly compared to other algorithms.

7 References

[1] Y. LeCun, C. Cortes, and C.J.C. Burges, "The MNIST database of handwritten digits," 1999. [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: May 12, 2023].